HUMBOLDT-UNIVERSITÄT ZU BERLIN

Techniken für installationsfreie Client-Software am Beispiel einer Versionsverwaltung von Simulationsmodellen

Diplomarbeit zur Erlangung des akademischen Grades

DIPLOMINFORMATIKER

eingereicht am Institut für Informatik
der Mathematisch-Naturwissenschaftlichen Fakultät II

von . . . . . Robert Schumann
geb. am 23.02.1982
in . . . . . Berlin

In Zusammenarbeit mit dem
Deutschen Zentrum für Luft- und Raumfahrt e.V.
Simulations- und Softwaretechnik

Gutachter:	Prof. Dr. Jens-Peter Redlich
	Dr. Andreas Gerndt
Betreuer:	Dipl.-Ing. Daniel Lüdtke
eingereicht am:	1. Juni 2012

Inhaltsverzeichnis

Abkürzungsverzeichnis

ABI	Application Binary Interface
AIR	Adobe Integrated Runtime
AJAX	Asynchronous JavaScript and XML
ANSI	American National Standards Institute
API	Application Programming Interface
CERN	Conseil Européen pour la Recherche Nucléaire
CIL	Common Intermediate Language
CLR	Common Language Runtime
COM	Component Object Model
CSS	Cascading Style Sheets
CVS	Concurrent Versions System
GPL	GNU General Public License
HTML	Hypertext Markup Language
HTML5	Hypertext Markup Language Version 5
HTTP	Hypertext Transfer Protocol
HTTPS	Hypertext Transfer Protocol Secure
JDK	Java Development Kit
JNLP	Java Network Launching Protocol
JRE	Java Runtime Environment
JSON	JavaScript Object Notation
JVM	Java Virtual Machine
LTE	Long Term Evolution
MVC	Model View Controller
NAT	Network Address Translation

NFS	Network File System
PHP	PHP Hypertext Preprocessor
RDP	Remote Desktop Protocol
REST	Representational State Transfer
RIA	Rich Internet Application
SimMoLib	Simulation Model Library
SSH	Secure Shell
UMTS	Universal Mobile Telecommunications System
URI	Unique Resource Identifier
W3C	World Wide Web Consortium
WebDAV	Web Distributed Authoring and Versioning
WebGL	Web Graphics Library
WHATWG	Web Hypertext Application Technology Working Group
XHTML	Extensible Hypertext Markup Language
XML	Extensible Markup Language
XPI	Cross-Platform Install
XUL	XML User Interface Language

1 Einleitung

Das Internet und die damit einhergehende Vernetzung von unzähligen Rechnersystemen verändern die Art der Nutzung von Software derzeit stark. Dabei hat sich eine vielseitig ausgeprägte Client-Server-Infrastruktur entwickelt, in der ein Rechner als Server einem anderen Rechner als Client einen bestimmten Dienst, eine Funktionalität oder eine Anwendung über das Netzwerk bereitstellt. Die Fähigkeit des Clients, sich sofort beim Systemstart mit dem Netzwerk zu verbinden, ermöglicht sogar den Bezug des Betriebssystems über das Netzwerk. Damit wird lokal installierte Software und deren Wartung auf dem Client in solchen Spezialfällen bereits überflüssig. Ist das Betriebssystem einmal gestartet, kann der Client mit der darin vorhandenen Software wiederum Server-Dienste oder servergestützte Anwendungen in Anspruch nehmen. Neue Prinzipien wie *Software as a Service* [Bux08] und *Cloud Computing* [Bau10] ermöglichen es Anwendern, solche Art von Diensten mit geringem Installationsaufwand und je nach Bedarf zu nutzen. Rechenzeit und Software werden über die Netzwerkverbindung auf beziehungsweise von einem Server zur Verfügung gestellt. Der Anwender benötigt lediglich einen Netzwerkzugang und eine passende Schnittstelle zum Dienst, die zum Beispiel als Anwendung innerhalb eines Webbrowsers bereitgestellt werden kann.

E-Mail-Anwendungen, mit denen der Nutzer im Webbrowser E-Mails empfangen, senden oder archivieren kann, gehören beispielsweise zu diesen Diensten, die als Software im Webbrowser realisiert und angeboten werden. Die notwendige Kommunikation mit den beteiligten Mail-Servern übernimmt hierbei der Web-Server, mit dem sich der Webbrowser verbindet. Der Nutzer selbst interagiert lediglich mit der grafischen Oberfläche der Anwendung. Er tauscht Daten mit dem Web-Server aus, der seinerseits die Funktionen der Mail-Server nutzt und für den Nutzer transparent macht. Das Prinzip der hier vorliegenden Abstraktion lässt sich auf beliebige andere Dienste ebenso anwenden und wurde bereits in [Tur99] genauer klassifiziert. Der besondere Vorteil dieser Art von Software liegt in der sofortigen Ausführung innerhalb des Webbrowsers nach ihrem Herunterladen in den Zwischenspeicher des Webbrowsers. Eine notwendige Installation wie bei klassischer Software entfällt.

Der Prozess der Installation einer Software umfasst ihr Kopieren in den Speicherbereich eines Betriebssystems, in welchem Daten aufbewahrt werden, und ihre eventuelle

Anpassung an das Betriebssystem, um sie dort für den Nutzer verfügbar zu machen [8]. In vielen Fällen ist der Speicherbereich das Dateisystem und die Software bleibt dort unverändert erhalten, da es sich um nichtflüchtigen Speicher handelt. Zusätzlich können bei der Installation systemrelevante Änderungen auftreten, die normalerweise durch einen privilegierten Nutzer des Betriebssystems ausgeführt werden müssen. Die Komplexität eines solchen Vorgangs wird oft durch ein zusätzliches Programm gekapselt, welches als Installer bezeichnet wird und vom Nutzer ausgeführt werden muss.

Im Vergleich mit der zuvor beschriebenen netzwerkgestützten Client-Server-Infrastruktur ist es daher motivierend, Techniken für Client-Software genauer zu untersuchen, die nicht explizit installiert werden muss. Eine Evaluation ihrer Fähigkeiten ist in diesem Zusammenhang eine sinnvolle Bewertungsgrundlage. Für bestimmte Anwendungsfälle wie die erwähnte E-Mail-Anwendung scheinen die Techniken bereits in der Lage zu sein, klassisch installierte Software vom Funktionsumfang her annähernd zu ersetzen. Einerseits entfällt jegliche direkte oder indirekte Manipulation des Client-Betriebssystems durch Installationen bei der Nutzung solcher Software. Andererseits wird auch das Problem von notwendigen Updates beseitigt, da der Server über das Netzwerk jederzeit die aktuelle Version der Software an den Client ausliefert. Privilegierte Nutzerrechte sind außerdem in den meisten Fällen zur Ausführung nicht notwendig oder durch systembedingte Sperren gar nicht erst möglich. Das ist prinzipiell ein Vorteil gegenüber dem beschriebenen herkömmlichen Mechanismus einer Installation. Mögliche Plattformen für *installationsfreie Client-Software* und deren Eigenschaften werden daher in den folgenden Kapiteln untersucht.

Anhand des konkreten Beispiels und eines zu erstellenden Demonstrators wird in dieser Arbeit des Weiteren untersucht, ob und inwiefern die Nutzung einer Versionsverwaltung mittels installationsfreier Client-Software möglich ist. Eine Versionsverwaltung kann als abstraktes Speichersystem betrachtet werden, welches alle Änderungen in Dateien und Ordnern erkennt, erfasst und in einer Versionsgeschichte zugänglich macht. Darüber hinaus ist die Versionsverwaltung in der Regel im Netzwerk erreichbar, so dass die enthaltenen Daten von den Netzwerkteilnehmern jederzeit abgerufen werden können. Ferner können sie durch Bearbeitung der Daten neue Versionen erstellen und zur Versionsverwaltung hinzufügen. Ein derartiges System ermöglicht einer größeren Nutzergruppe außerdem eine koordinierte Arbeit an gemeinsamen Projekten [14], ohne dabei ältere Versionen oder Fragmente von Dateien wegen Überschreibung aufgeben zu müssen.

1.1 Umfeld der Arbeit

Im Deutschen Zentrum für Luft- und Raumfahrt e.V. (DLR) arbeiten Akademiker und Ingenieure vieler Fachrichtungen interdisziplinär an der Entwicklung von computergestützten Simulationsmodellen technischer Geräte. Computersimulationen erhöhen die Zuverlässigkeit der aus den Modellen abgeleiteten realen Systeme und reduzieren Kosten, da der Verlust eines technischen Systems wirtschaftlich und menschlich katastrophal sein kann. Außerdem ist es sinnvoll, alle möglichen Fehlerursachen oder problematischen Veränderungen so simulieren zu können, dass deren Eintrittswahrscheinlichkeit von vornherein minimiert werden kann. Gerade aufgrund der umfangreichen Simulations- und Testphasen sind die Modelle ständigen Änderungen unterworfen.

In der Simulation Model Library (SimMoLib) werden die Simulationsmodelle gespeichert, verwaltet und zur Verfügung gestellt. Eine Versionsverwaltung im Hintergrund von SimMoLib ermöglicht außerdem das Nachverfolgen der Änderungen, das Aufteilen und die Kennzeichnung einer bestimmten Version des Simulationsmodells, sowie viele andere Maßnahmen zur Bewältigung der als Dateien vorliegenden Modelle. SimMoLib versteht sich als Software-Bibliothek für den Anwenderkreis, der beim DLR derzeit hauptsächlich aus Raumfahrttechnikern, Ingenieuren und Physikern zusammengesetzt ist. Eine weiterführende Betrachtung der kollaborativen Entwicklung von Simulationsmodellen für Raumfahrtsysteme im Rahmen von SimMoLib wird in [LAD+12] vorgenommen.

Abbildung 1.1 zeigt schematisch den Aufbau von SimMoLib. Ein Server für die Versionsverwaltung (SVN-Server) und ein Server für den einfachen Zugriff auf die Modelle über eine Website (CouchDB-Server) bilden die Grundlage, auf denen der SimMoLib-Server aufsetzt. Durch Schnittstellen mit diesen beiden Servern realisiert er die Funktionalität der Bibliothek für Simulationsmodelle, indem er neue Modelle anlegt oder Daten für die Website zur Verfügung stellt. Der SimMoLib-Client kommuniziert je nach Aufgabe mit einem der Server. Er überträgt beispielsweise Änderungen an Modellen direkt in das Versionsverwaltungssystem, wohingegen er die Veröffentlichung einer dedizierten Version als sogenanntes Release beim SimMoLib-Server anfordert. Als Releases werden in der Regel stabile Versionen von Software bezeichnet, die Nutzern zugänglich gemacht werden sollen.

Als Konsolidierung und Vereinfachung dieses Aufbaus wäre eine Zusammenlegung von SimMoLib-Server und -Client denkbar, da die Funktionen innerhalb der stilisierten Wolke in Abbildung 1.1 auch von einer alleinstehenden Client-Software ausgeführt werden könnten. Die in dieser Arbeit vorgenommene Auswertung von verfügbaren

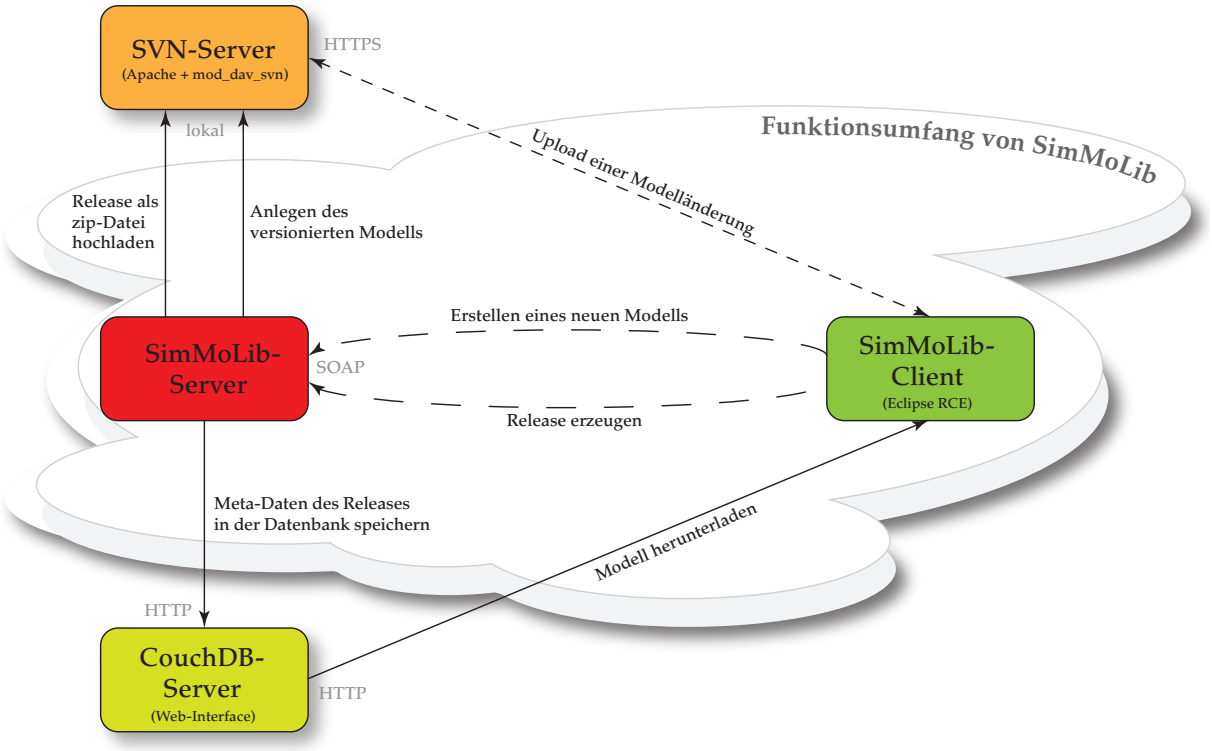

Abbildung 1.1: Aufbau der Bibliothek SimMoLib beim DLR

Techniken für installationsfreie Client-Software ist daher auch im Hinblick auf mögliche technische Änderungen bei SimMoLib als sinnvoll zu betrachten. Dennoch steht die Art der Konfiguration stellvertretend für viele Client-Server-Architekturen, bei denen ein oder mehrere Server gewisse Funktionalitäten für den Client übernehmen, die dieser nicht selbst bereitstellen kann.

Die Anwendung und korrekte Benutzung einer Versionsverwaltung erfordert informationstechnische Kenntnisse, welche bei der Zielgruppe der Mitarbeiter des DLR noch nicht vollständig vorausgesetzt werden können. Derzeit wird der schreibende Zugriff auf SimMoLib über einen eigens entwickelten javabasierten Client bereitgestellt, welcher zunächst heruntergeladen und dann lokal unter Windows ausgeführt werden muss. Der Client basiert auf einer funktional umfangreichen Rich-Client-Plattform, welche mittels Eclipse realisiert wird [47]. Eclipse ist eine modular anpassbare und grafische Programmierumgebung. Sie dient in den meisten Fällen als Frontend für Programmierer und nicht wie bei SimMoLib direkt als Programm für den Endnutzer, kann aufgrund ihrer Fähigkeiten und Möglichkeiten zur Modifikation allerdings auch als interaktive Nutzeroberfläche gestaltet werden. Für die Anwender wird der Client visuell angepasst

und für die Aufgabe des Zugriffs auf die Simulationsmodelle innerhalb von SimMoLib reduziert.

Die eigentliche Versionsverwaltung der Modelldateien geschieht derzeit auf einem Subversion-Server. Subversion ist ein zentrales Versionsverwaltungssystem der Apache Foundation und weit verbreitet. Andere Versionsverwaltungen wie Mercurial oder das auf lokale Code-Bäume optimierte git sind ebenso denkbar und wurden in [Ker10] untersucht. Die Anforderungen an eine zentrale Bibliothek mit einem einfachen Zugriff auf Simulationsmodelle wurden von Subversion allerdings bereits erfüllt, so dass zum jetzigen Zeitpunkt weitere Effizienzbetrachtungen im Rahmen von SimMoLib nicht vorgenommen werden müssen. Von Interesse ist jedoch die Einbeziehung von installationsfreier Client-Software zur Realisierung des Zugriffs auf SimMoLib beziehungsweise die Erforschung, inwiefern die lokale Installation des javabasierten Clients nach wie vor notwendig ist. Da SimMoLib ohnehin als zentrale Versionsverwaltung für eine Client-Server-Infrastruktur ausgelegt und ein Web-Interface bereits vorhanden ist, stellt sich die Frage nach der Möglichkeit der Umsetzung eines vollständig funktionsfähigen Web-Clients auf Basis von Techniken für installationsfreie Client-Software.

1.2 Aufgabenstellung

Die steigende Bedeutung von Diensten im Netzwerk und der damit zusammenhängenden Client-Software, sowie die stetig wachsende Nutzeranzahl [26] führen zu einer raschen Weiterentwicklung der diesbezüglich angewandten Techniken. Sogenannte Rich Internet Applications (RIA), die als Anwendungen innerhalb des Webbrowsers ausgeführt werden, können Aufgaben und Funktionen von Software übernehmen, die bisher klassisch als alleinstehende Anwendung installiert wurde. Die RIAs arbeiten dabei oft als Client mit einem Server als korrespondierendem Gegenstück zusammen, welcher Daten speichert oder technische Aufgaben übernimmt. Die Verknüpfungen zwischen Server und Client sind hierbei sehr vielfältig. Die Machbarkeit einer Anwendung definiert sich jedoch in den meisten Fällen über die Fähigkeiten der Client-Software. Eine Betrachtung von diesbezüglicher Server-Software wurde beispielsweise in [Tur99] vorgenommen.

Die Möglichkeit der Nutzung einer Versionsverwaltung, welche im Rahmen des Vorhabens SimMoLib beim DLR evaluiert wird, liefert ein praktisches Beispiel zur genaueren Untersuchung der Techniken für installationsfreie Client-Software. Ein prototypisch erstellter Demonstrator soll Vorteile und Nachteile einer solchen Realisierung aufzeigen und eine Grundlage der Machbarkeitsabschätzung für das DLR sein. Essentiell für eine

Versionsverwaltung sind Kommunikations- und Speicherschnittstellen auf dem Client, um Daten transportieren und sichern zu können. Diesbezüglich bestehende Einschränkungen bei den hier vorgestellten Techniken werden im Besonderen betrachtet und neue Lösungswege und Anwendungsgebiete untersucht. Dabei steht ein systematisches Konzept zur Absicherung bei der Ausführung installationsfreier Client-Software im Mittelpunkt, das als *Sandbox* (oder auch Sandkasten) bezeichnet wird.

Eine installationsfreie Client-Software für eine Versionsverwaltung ohne zusätzlich notwendige Server-Software existiert aufgrund dieser Beschränkungen nach Wissen des Autors derzeit nicht. Es besteht jedoch die Möglichkeit, eine solche mit Hilfe neuester Entwicklungen unter Einbeziehung von Webtechnologien zu erforschen. Unter Webtechnologien soll im Folgenden die Art von softwaretechnischen Lösungen verstanden werden, welche durch einen Webbrowser bereitgestellt werden können. Die Motivation hierfür liegt in der plausiblen Annahme, dass heutzutage viele vom Menschen genutzte Rechnersysteme über einen Webbrowser verfügen. Spezielle Anwendungsfälle wie bei Embedded-Geräten verlangen eventuell nach anderen Lösungen und werden im Rahmen dieser Arbeit nicht weiter betrachtet.

1.3 Struktur der Arbeit

Zunächst wird im Kapitel 2 ein Überblick des aktuellen Stands der Technik im Bezug auf Client-Software im Allgemeinen gegeben. Dies ist unabdingbar, um Aussagen zur Installationsfreiheit von Software überhaupt treffen zu können. Unterschiedlichste Konzepte zur Realisierung von Anwendungen auf einem Client-Rechner werden gegenübergestellt und installationsfreie Client-Software im Rahmen dessen näher spezifiziert.

Im folgenden Kapitel 3 wird anhand verschiedener Kriterien eine Evaluation möglicher Techniken vorgenommen, die installationsfreie Client-Software realisieren könnten. Dabei stehen vor allem grundsätzliche Eigenschaften von Software, Einschränkungen und Sicherheitsaspekte im Vordergrund. Spezifische Anforderungen im Hinblick auf das hier zu behandelnde Beispiel werden analysiert und untersucht.

Im Fokus von Kapitel 4 steht das Beispiel eines Web-Clients für den Zugriff auf eine Versionsverwaltung mittels Webtechnologien. Dazu werden im Rahmen eines Prototypentwurfs die einzelnen zuvor geforderten Kriterien installationsfreier Client-Software auf ihre Realisierbarkeit hin untersucht und falls möglich umgesetzt. Eine Machbarkeitsabschätzung für den Produktiveinsatz beim DLR bietet Anhaltspunkte für eine Beurteilung zur realen Umsetzung.

Schließlich gewährt Kapitel 5 nach einer Zusammenfassung der vorgestellten Techniken einen Ausblick auf die weitere Entwicklung im Bereich der hier betrachteten Software und deren Anwendungsgebiete.

2 Stand der Technik

Jede Software erfordert eine entsprechende Laufzeitumgebung, in der sie ausgeführt oder interpretiert wird. Eine Laufzeitumgebung kann durch das Betriebssystem und die darin vorhandenen Programmbibliotheken selbst bereitgestellt werden. Des Weiteren existieren vom Betriebssystem abstrahierende Software-Plattformen, die ebenso in der Lage sind, die Ausführung von Anwendungen zu ermöglichen. Das erfordert in der Regel zunächst eine Installation der Software-Plattform im jeweiligen Betriebssystem. Anschließend können auf Basis der Software-Plattform erstellte Anwendungen ausgeführt werden. Das bedeutet im Einzelnen, dass sie nicht mehr an die Techniken des Betriebssystems wie die Programmiersprache gebunden sind, sondern nur noch an die der Software-Plattform, die vom Betriebssystem abstrahiert und eine zusätzliche Laufzeitumgebung bildet [Fie00]. Als eine simple Abstraktionsebene kann beispielsweise eine Kommando-Shell angesehen werden, in der Befehle interpretiert werden, um ein Programm zu realisieren [SG98]. Die Befehle der Shell stehen nicht im direkten Zusammenhang mit den eigentlichen Betriebssystem-Funktionen, die schlussendlich die Funktionalität des Programms realisieren.

Die Vernetzung von Rechnersystemen führt zusätzlich zu einer Aufgabenteilung zwischen den Netzwerkteilnehmern, wobei ein System eine bestimmte Funktion wie einen Dienst bereitstellt und ein anderes diesen über die Kommunikation im Netzwerk nutzt. Im Folgenden werden daher Techniken vorgestellt, die auf einem Rechner und innerhalb einer Client-Server-Architektur verfügbar beziehungsweise anwendbar sind. Der Begriff *installationsfrei* ist ein zentraler Bestandteil der Betrachtungen in dieser Arbeit. Mangels einer allgemein anerkannten Definition für *installationsfreie Client-Software*, erfolgt die Untersuchung der Techniken in der vorliegenden Arbeit unter der Prämisse, dass

- *installationsfrei* bedeutet, eine Anwendung oder Software unmittelbar ohne vorangegangenen Installationsvorgang wie in [8] beschrieben ausführen und nutzen zu können und
- *Client-Software* eine Software bezeichnet, die im Netzwerk eine Funktion als Client eines Server-Dienstes realisiert und auf dem Rechner ausgeführt wird, den der Nutzer zum Zugriff auf den Server-Dienst nutzt.

Die folgenden Abschnitte befassen sich mit einer Vorstellung von Software-Arten im Allgemeinen und Beispielen dafür, um installationsfreie Client-Software in diesem Gefüge einordnen zu können. Darüber hinaus wird kurz auf die Technik einer Versionsverwaltung eingegangen, welche als konkretes Beispiel für die Anwendung der Techniken für installationsfreie Client-Software dienen soll. Im Mittelpunkt dieser Betrachtungen steht die Einbettung der Software in das Netzwerk, da Client-Software nur innerhalb eines Netzwerks sinnvoll ist. Der Aufbau und die Funktionsweise der Client-Server-Infrastruktur wird daher ebenso erläutert. Eine Evaluation der Eigenschaften und Fähigkeiten derartiger Software folgt im nächsten Kapitel.

2.1 Das Netzwerk als Software-Lieferant und Dienstanbieter

Die Art und Weise Software zu beziehen, hat sich mit der Verfügbarkeit und Nutzung von Rechnernetzwerken kontinuierlich verändert. Zuvor wurde Software auf physikalischen Trägern wie Floppy-Disks, CDs, DVDs oder USB-Datenspeichern verteilt und von dort in das Dateisystem des Rechners kopiert. Im Gegensatz zum elektronischen Bezugsweg über das Netzwerk offenbaren sich hier zumindest zwei Nachteile. Einerseits wird die auf den physischen Datenträgern enthaltene Software-Version nach kurzer Zeit inaktuell und andererseits muss der Datenträger dem Rechner erst in irgendeiner Form zugänglich gemacht werden. Die Nutzung von Software aus dem Netzwerk hingegen bedarf oft lediglich einer minimalen Interaktion des Nutzers innerhalb einer grafischen Benutzeroberfläche. Außerdem wird die so bezogene Software in der Regel automatisch aktualisiert und kann infolgedessen leicht vom Entwickler erweitert und verbessert werden [Fie00]. Darüber hinaus haben sich Netzwerke, allen voran das Internet, auch als Quelle für direkt ausführbare Anwendungen etabliert, die beispielsweise im Webbrowser ausgeführt werden können und für die die genannten Vorteile ebenso gültig sind. Die Fähigkeiten von Webbrowsern haben sich im Zuge dessen bereits massiv weiterentwickelt, so dass Web-Anwendungen wie *acrobat.com* [1] oder *Microsoft Office 365* [35] zur Dokumentenverarbeitung lokal installierte Software teilweise ersetzen können.

Der Webbrowser stellt in vielen Fällen außerdem das Bindeglied für Software dar, die er nicht direkt ausführt, sondern nur herunterlädt und einer passenden Software-Plattform übergibt. Die Realisierung von Client-Software ist allerdings unabhängig von der Art ihres Bezugs oder ihrer Verteilung durch viele Software-Plattformen möglich.

Die Vernetzung führt jedoch auch zu einer Sicherheitsproblematik, da jede Netzwerkkommunikation eine potentielle Schwachstelle für die Sicherheit eines Systems sein

kann. Das gilt nicht nur für Schadcode, der empfangen und ausgeführt werden kann, sondern auch für mögliche Fehler in der Implementation der Kommunikation oder der Anwendungen selbst, die zu einem Absturz führen können. Eine bestehende Netzwerkverbindung kann in diesem Fall besonders schädlich sein, da Systeme durch unkontrollierte Abstürze oft angreifbarer werden. Firewalls, die eine schädliche Netzwerkkommunikation unterbinden und andere Sicherheitsmechanismen wie eine Begrenzung des Programm-Codes auf systemunkritische Funktionen können diese Gefahren zumindest teilweise beheben.

2.1.1 Client-Server-Architektur

Eine wesentliche Anwendung von Netzwerken neben der Kommunikation besteht in der Aufgabenteilung der Netzwerkteilnehmer. Server sind in der Regel abgesicherte, hochverfügbare und leistungsfähige Rechnersysteme, die ein zuverlässiges Arbeiten mit Diensten, abgesichertem externen Speicher und Software erst ermöglichen [Ber96]. In heute verbreiteten Netzwerken ist ein zugreifbarer Dienst auf einem Server durch eine Schnittstelle definiert, die als *Socket* bezeichnet wird. Der Socket eines Servers ist ein Endpunkt einer Interprozesskommunikation, der Socket eines Clients der dazugehörige andere [SFR04]. Zwischen diesen beiden Endpunkten werden Daten von zwei Prozessen ausgetauscht, die jeweils komplementär auf den beteiligten Rechnern ausgeführt werden. Im Fall des Internet-Protokolls (IP) wird der Socket über eine IP-Adresse und einen Port definiert. Die Kommunikation und der Datenaustausch findet mittels definierter Protokolle statt, die aufgrund der Vielfalt der Dienste im Netzwerk und auch geschuldet der vielen Anwendungsbereiche mannigfaltig sind. Das Anwendungsprotokoll Hypertext Transfer Protocol (HTTP) realisiert zum Beispiel den zustandslosen Abruf von Websites über das Netzwerk mittels Webbrowser, während das Secure Shell (SSH)-Protokoll eine dauerhafte Verbindung zwischen zwei Endpunkten herstellt, mit der simultan Daten wie die Eingabe und Ausgabe von Befehlen übertragen werden können. Die in dieser Arbeit betrachtete Versionsverwaltung folgt ebenfalls diesem Prinzip, indem ein Server verschiedene Versionen von Dateien über ein Kommunikations- und Anwendungsprotokoll für einen Client vorhält.

Die Verfügbarkeit von Server-Diensten hängt maßgeblich von der Netzwerkkonfiguration ab, die für die beteiligten Rechner gilt. Netzwerke lassen sich logisch in lokal und global zusammenhängende Einheiten einteilen [43] und es existieren Protokolle wie die Network Address Translation (NAT) [49] für die korrekte Übersetzung der Kommunikation zwischen ihnen. Die Gesamtheit der Client-Software in einer Client-Server-Architektur bildet keine einheitliche Struktur. Sie ist vielmehr durch gewisse

Einschränkungen auf dem Client und technische Spezialfälle evolutionär gewachsen und dem jeweiligen Zweck angepasst. Der Funktionsumfang von Server und Client variiert stark je nach Anforderung des Nutzungsszenarios. In vielen Fällen ist der Client auch selbst ein Server, der wiederum anderen Clients Daten oder einen Dienst zur Verfügung stellt.

2.1.2 Betriebssysteme über PXE

Etwas abseits von Client-Software im *klassischen* Sinn ist das Preboot Execution Environment (PXE) [27] angesiedelt. Es schafft eine Möglichkeit zur Programmausführung bevor das eigentliche Betriebssystem des Rechners gestartet wurde. Realisiert wird dies über eine PXE-fähige Netzwerkkarte, die beim Start des Rechners noch vor dem Booten eine Netzwerkadresse und über ein simples Dateiübertragungsprotokoll ein vom Rechner ausführbares Programm bezieht. In der Regel ist dieses Programm ein Bootloader und besteht aus minimalen Routinen, um den Start des Rechners beeinflussen zu können. Im einfachsten Fall erkennt der Bootloader wie zum Beispiel iPXE [9] die angeschlossenen startfähigen Speichermedien oder lädt ein Betriebssystem über das Netzwerk nach. Für Letzteres bietet sich oft ein Linux-Betriebssystem an, da es prinzipbedingt nur aus einem kleinen Kern und einem initialen Dateisystem bestehen muss. Der Kern unterstützt in kompakter Form die Hardware des Rechners und das Dateisystem enthält die im Betriebssystem auszuführenden Programme und die logische Struktur der enthaltenen Daten [Kof07]. Auf diese Weise kann eine minimale Betriebssystemsoftware für Clients verteilt werden, die direkt vom Netzwerk auf den Client geladen und dort ausgeführt wird. Der Wartungsaufwand für lokale Installationen kann somit entfallen. In größeren Firmennetzwerken dient diese Technik beispielsweise als Basis für eine einheitliche softwaregestützte Client-Server-Architektur. In heterogenen Netzwerken oder dem Internet ist diese Technik bislang größtenteils irrelevant, da zu ihrer Nutzung einige technische Voraussetzungen im Netzwerk notwendig sind. Bei herkömmlichen heimischen Internetanschlüssen steht beispielsweise die Einwahl der DSL-Verbindung dieser Technik im Weg und auch die vom Provider zugewiesene IP-Adressen-Information müsste weitere Optionen enthalten.

2.1.3 Software aus der Cloud

Neben Software-Plattformen für Anwendungen auf dem Client existieren mittlerweile sogenannte Cloud-Dienste im Netzwerk. Das englische Wort für *Wolke* suggeriert eine gewisse Transparenz und Undefinierbarkeit in der Art, wie Dienste und Anwendungen

im Netzwerk verteilt werden und zugreifbar sind. Der Nutzer einer Anwendung hat über deren Realisierung und Bereitstellung in der Regel keine Information, er muss sich nur entscheiden, ob und in welchem Umfang er sie nutzen möchte. Eine exzessive Nutzung führt zu steigenden Kosten, eine geringe oder minimale Nutzung ist oft kostenfrei möglich. Der Dienst und die Anwendung werden so je nach Bedarf mit Hilfe von Metriken verteilt. Die Hardware und Software des Rechenzentrums, die diesen Dienst als indifferente Wolke skalierbar realisieren, werden als Cloud bezeichnet [AFG+10]. Die Synergie aus Client-Software und Server-Dienst in der Cloud ermöglicht nahezu alle Arten von Anwendungen, da fehlende Funktionalität auf dem Client durch den Server ausgeglichen werden kann. Seit der ständig verfügbaren Internet-Verbindung über die Funkstandards Universal Mobile Telecommunications System (UMTS) oder Long Term Evolution (LTE) ist diese Form Bereitstellung von Software und Diensten vor allem für mobile Geräte wie Smartphones oder Tablet-PCs wegen ihrer reduzierten Kapazität und Leistungsfähigkeit interessant. Es existieren allerdings auch Ansätze für modular aufgebaute Betriebssysteme, die lediglich das absolute Minimum für den Betriebssystemstart installieren. Sämtliche Programme werden dann während der Laufzeit aus der Cloud in den flüchtigen Arbeitsspeicher geladen und bleiben nur bis zum nächsten Neustart erhalten [48].

2.1.4 Server-basierte Applikationen

Server-basierte Anwendungen ermöglichen dem Client über eine Schnittstelle wie dem Webbrowser maximale Funktionalität bei minimalen Anforderungen auf Client-Seite. Verstärkt wurde die Entwicklung derartiger Anwendungen durch die große Verbreitung von Webbrowsern aber gleichzeitig strikte Beschränkung derer auf statische Inhalte. Neben server-basierten Konzepten für komplette Software-Plattformen wie *Java Servlets*, *Ruby on Rails* oder dem auf Python basierenden *Django*, sind klassische Skriptsprachen wie Shell, Perl oder PHP Hypertext Preprocessor (PHP) immer noch wichtige Komponenten auf Webservern, um Inhalte je nach Anwendungszweck ausliefern können. Es können beispielsweise Datenbanken angesprochen oder gesendete Formulardaten verarbeitet werden. Mit den stetigen Erweiterungen von Webbrowser-Fähigkeiten auf dem Client und dazu kompatiblen Anwendungen auf dem Server wurden manche Server-Konzepte jedoch bereits überflüssig. CouchDB ist zum Beispiel eine dokumentenorientierte Datenbank, mit der Client-Software über HTTP ohne den Zwischenschritt einer Server-Anwendung in vollem Umfang kommunizieren kann. Server-basierte Techniken sind im Folgenden nicht weiter relevant für die Vorstellung und Auswertung installationsfreier Client-Software, daher wird an dieser Stelle auf weiterführende

Betrachtungen in [Tur99] und [Bau10] verwiesen.

2.2 Betriebssystemabhängige Client-Software

Client-Software für Dienste im Netzwerk ist seit der Entstehung von Computer-Netzwerken Ende der 1960er Jahren direkt in Betriebssysteme eingeflossen. Die Verfügbarkeit von Rechenzeit, Telekommunikation und Terminalprogrammen ermöglichte die Nutzung eines Servers durch ein angebundenes Terminal. Die Anwendungsbereiche haben sich seitdem ständig erweitert, so dass auch die Fähigkeiten einer Client-Software erweitert werden mussten. Betriebssystemabhängige Client-Software funktioniert in der Regel durch Bindung an die Programm-Bibliotheken des Betriebssystems, die verschiedene Funktionen für Anwendungen bereitstellen. Diese Bindung erfolgt beispielsweise dynamisch über Programme wie einen *Linker*, der vereinfacht gesagt alle Funktionen innerhalb des Betriebssystems kennt und der Client-Software die Adresse des zur Laufzeit benötigten Programmcodes im Speicher nennen kann. Die Anwendung springt dann bei Bedarf zu dieser Adresse und führt den Code im eigenen Kontext aus [17].

2.2.1 Native Client-Software

Die Bedeutung von Client-Software im Allgemeinen ist auch bei der Erstellung und Nutzung von Betriebssystemen relevant. Häufig wird zusammen mit dem Betriebssystem nativ unterstützte Client-Software für den Zugriff auf Netzwerkdienste verteilt. Die Bandbreite reicht hier von Anwendungen für entfernte Kommandozeilen wie SSH über entfernte Desktops bis hin zu Netzwerkspeicher. Unix- und Linux-Betriebssysteme unterstützen beispielsweise in der Regel nativ eine Datenübertragung von einem Server durch das Einbinden von Netzwerkspeicher über das Network File System (NFS) genannte Protokoll. Die Betriebssysteme von Microsoft für Desktop-PCs unterstützen von Haus aus eine Desktop-Verbindung zu einem entfernten Windows-Rechner über das Remote Desktop Protocol (RDP). Auch wenn Teil-Portierungen dieser Protokolle auf die jeweils andere Betriebssystemplattform existieren, sind die Implementationen und ihre Anwendung oft nicht einheitlich und funktionell unterschiedlich. Teilweise existiert die Software auch nur entweder für den Client oder den Server, also lediglich für einen Endpunkt der Kommunikation.

2.2.2 Externe Client-Software

Falls eine betriebssystemabhängige Client-Software nicht nativ in das Betriebssystem integriert ist, erfordert sie in den meisten Fällen eine korrekte Installation und eine explizite Abstimmung auf das Betriebssystem. Das gilt eingeschränkt auch für sogenannte *Portable Apps*, die zwar nicht installiert werden müssen, aber dennoch betriebssystemspezifisch sind. Versionsunterschiede der Programm-Bibliotheken oder der Betriebssysteme führen bei dieser Art von Software häufig zu Fehlern bei der Ausführung oder verhindern diese gänzlich. Zur Vermeidung solcher Probleme gibt es Konzepte wie das Application Binary Interface (ABI), welches zum Ziel hat, Binärkompatibilität auch über Versionen von Programmbibliotheken hinweg zu garantieren. Es ist jedoch in der Regel nicht anwendbar bei verschiedenen Rechnerarchitekturen.

2.2.3 Beispiele

Heutzutage verbreitete Betriebssysteme wie Windows, Linux und OS X und deren Betriebssystembibliotheken werden größtenteils mit den Programmiersprachen C und den objektorientierten Weiterentwicklungen wie C++ programmiert. Die entsprechende Client-Software muss ebenfalls damit umgesetzt werden, wenn sie die Funktionen des Betriebssystems nutzen können soll. Andere Programmierumgebungen sind denkbar und wurden wie Smalltalk auch tatsächlich eingesetzt, sie scheiterten jedoch oft an mangelnder Effizienz und mangelnder Leistungsfähigkeit der Rechner selbst. Für den Menschen einfach zu benutzende und abstrakte Programmiersprachen oder gar direkt ausführbare Skriptsprachen erfordern eine höhere Rechenkapazität, da die menschlich verständlichere Abstraktion die Komplexität für den Rechner erhöht. Beispielsweise existieren in vollständig objektorientierten Programmiersprachen tatsächlich nur Objekte mit Eigenschaften und Methoden. Diese lassen sich aber nicht ohne Weiteres in einem Speicherregister ablegen, sondern müssen erst in eine Datenstruktur umgewandelt und interpretiert werden. Hardwarenahe Programmiersprachen schreiben hingegen direkt Werte in ein Speicherregister und arbeiten zudem mit der vorhandenen Prozessorarithmetik, wodurch sie im Allgemeinen sehr effizient auf einem Rechner genutzt werden können.

Smalltalk

Ab Ende der 1960er Jahre wurde im Xerox PARC-Forschungszentrum in Kalifornien die dynamische, strikt objektorientierte und untypisierte Programmiersprache und Entwick-

lungsumgebung Smalltalk von Alan Kay und anderen entwickelt [Kay96]. Die dabei verwendeten Methoden und Ideen wie grafische Benutzeroberflächen, Klassen, Vererbung, Nachrichten zwischen Objekten beeinflussten ganze Generationen nachfolgender objektorientierter Programmiersprachen. Das von Smalltalk eingeführte Model View Controller (MVC)-Prinzip [KP88] oder Abwandlungen davon haben sich auch bei der Entwicklung von Web-Anwendungen etabliert. Smalltalk enthält eine eigene virtuelle Maschine, die die primitiven Befehle der jeweiligen Rechnerarchitektur zur Umsetzung der Smalltalk-Programme nutzt. Im Gegensatz dazu verwendet der Programmierer lediglich Klassen, Methoden und Botschaften zur Realisierung einer Funktionalität. Dynamisch ist Smalltalk deswegen, weil die in Programmen enthaltenen Code-Blöcke erst zur Laufzeit ausgewertet werden und nicht bereits bei deren Übersetzung. Zur Entwicklungszeit von Smalltalk existierten jedoch kaum Rechner, die leistungsfähig genug waren, diesen hohen Aufwand an Abstraktion technisch umzusetzen. Das begünstigte die Verbreitung von Programmiersprachen wie C, die statisch, minimal strukturiert und prozedural arbeiten und damit leichter effizient umgesetzt werden können.

C / C++ / Objective-C

Die Programmiersprache C wurde in den 1970er Jahren von Dennis Ritchie entwickelt und 1978 erstmals von ihm und Brian Kernighan dokumentiert [KR78]. Sie bildete die Grundlage für das Betriebssystem UNIX 5 und ist auch heute noch nach Standardisierungen wie C89 und C99 des American National Standards Institute (ANSI) in den Jahren 1989 und 1999 vor allem wegen ihrer Effizienz eine der einflussreichsten Programmiersprachen für Betriebssysteme und Software. Sie ist außerdem eine echte Teilmenge der objektorientierten Weiterentwicklung C++, welche beispielsweise die Grundlage für moderne Windows-Betriebssysteme ist [Sch01]. Zur Erstellung von C- und C++-Programmen ist ein Compiler notwendig, der den Quelltext in Maschinencode übersetzt, dabei dem jeweiligen Rechnertyp anpasst und auch in der Lage ist, sie darauf zu optimieren. Beispielsweise kann der C-Compiler bestimmte Sprachkonstrukte und Befehle in spezielle Prozessorinstruktionen übersetzen, die in einer älteren Prozessorversion nicht vorhanden waren, um die Geschwindigkeit der Operationen zu erhöhen. In Anlehnung an Smalltalk wurde in den 1980er Jahren ein weiteres C-Derivat namens Objective-C von Brad Cox und Tom Love entwickelt [BW09]. Objective-C vereint die Smalltalk-Syntax mit einer Verbesserung des Laufzeitverhaltens durch Weglassen der dynamischen Code-Blöcke. Objective-C wird heute hauptsächlich für die Programmierung von Software unter OS X eingesetzt.

2.3 Abstraktion vom Rechner- und Betriebssystem

Betriebssystemabhängige Client-Software besitzt Einschränkungen hinsichtlich der Portabilität. Deshalb haben sich im Laufe der Zeit Software-Plattformen entwickelt, die für unterschiedlichste Betriebssysteme portiert werden können. Der Hersteller der portablen Software-Plattform stellt dabei sicher, dass die für die Plattform geschriebenen Programme auf allen Betriebssystemen die gleiche Funktionalität aufweisen, indem die Laufzeitumgebung der Software-Plattform die *eigene* Programmiersprache interpretiert und in die Programmiersprache des vorhandenen Betriebssystems übersetzt. Technisch gesehen unterscheiden sich Software-Plattformen oft durch bestimmte Programmierparadigmen wie zum Beispiel die Typisierung von Variablen oder Objektorientierung.

2.3.1 Schematische Darstellung von Abstraktionsebenen

Abbildung 2.1 veranschaulicht exemplarisch ein Modell von Abstraktionsebenen für Client-Software. Der Befehlssatz des Prozessors definiert die rudimentären Fähigkeiten und Möglichkeiten eines Rechners. Maschinencode besteht in der Regel aus einer sequenziell abzuarbeitenden Liste von Instruktionen, mit denen der Prozessor beispielsweise Speicherregister füllt, Additionen durchführt oder an bestimmte Stellen im Speicher springt, um dort weiterzuarbeiten. Während Maschinencode direkt vom Prozessor verarbeitet werden kann, muss eine höhere Programmiersprache erst durch einen Compiler in Maschinencode übersetzt werden. Solche höheren Programmiersprachen sind zahlreich vorhanden, unter ihnen beispielsweise Fortran und Haskell, die vor allem im wissenschaftlichen Bereich genutzt werden. C und dessen Erweiterungen sind allerdings bei der Anwendungsentwicklung für herkömmliche Server- und Endnutzer-Betriebssysteme so verbreitet, dass es zweifelsfrei als Stellvertreter für diese Art von Programmiersprachen in der vorliegenden Arbeit herangezogen kann. Die Programmierumgebung von C ist durch den Compiler und die Bibliotheken oft direkt mit dem Betriebssystem verbunden, so dass in C geschriebene Programme für Betriebssystem 1 nicht ohne Weiteres im Betriebssystem 2 funktionieren. Es ist aber möglich, in C oder C++ eigenständige Software-Plattformen zu erstellen, die ihrerseits eine portable Funktionalität, Programmierumgebung und Programmiersprache anbieten. Dies ist zum Beispiel bei Java der Fall, wo die Java Virtual Machine (JVM) in gewissen Grenzen für jede Kombination aus Betriebssystem und Rechnertyp explizit in C erstellt werden muss, wohingegen die Java-Programme selbst dann auf allen unterstützten Kombinationen gleich ablaufen. Das Ende der Abstraktionskette bilden Skriptsprachen

wie JavaScript oder Shell, die von dem Rechner und seiner Architektur komplett abstrahieren. Hier ist Speicher beispielsweise nur noch logisch in Variablen verfügbar und die Befehle werden komfortabel ohne notwendigen Übersetzungsschritt interpretiert.

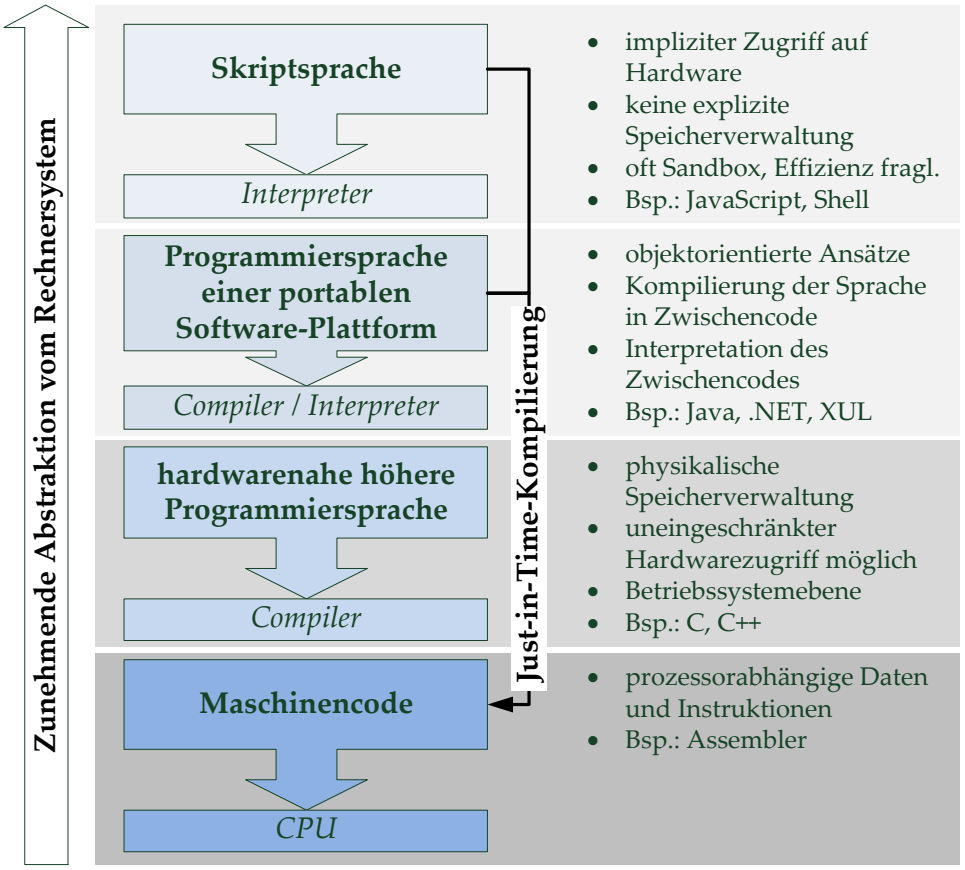

Abbildung 2.1: Exemplarische Abstraktionsebenen von Client-Software

2.3.2 Optimierung durch Just-in-Time-Compiler

Algorithmen arbeiten in Maschinencode oft um Größenordnungen schneller als in interpretierten Skriptsprachen wie JavaScript oder Shell, da sämtliche Zwischenschritte von Übersetzungen zwischen den Ausführungsplattformen entfallen. Mit zunehmender Komplexität von skriptbasierten Programmen wird die Geschwindigkeit daher zum Problem. Als Lösungen hierfür werden spezielle Just-in-Time-Compiler für Skriptsprachen wie JavaScript eingesetzt. Diese auch *Engines* genannten JIT-Compiler wie V8 von Google [20] oder TraceMonkey von Mozilla [36], die JavaScript nicht direkt interpretieren, sondern zur Laufzeit in Maschinencode übersetzen und dann ausführen, beseitigen die Laufzeitnachteile von stark abstrahierenden interpretierten Skriptsprachen fast voll-

ständig und bewahren dennoch den Komfort, den diese bieten. Auch Java nutzt die Technik eines JIT-Compilers situationsabhängig, um die Ausführungsgeschwindigkeit zu erhöhen. Allerdings wird hier nicht der Quelltext des Java-Programms zur Laufzeit in Maschinencode übersetzt, sondern ein Zwischencode, der zuvor schon bei der Erstellung des Java-Programms generiert wurde.

2.4 Portable Plattformen für Client-Software

Portable Software-Plattformen abstrahieren von den eigentlichen Betriebssystemfunktionen und -bibliotheken. Sie definieren eine eigene Programmiersprache, mit der der Programmierer die gewünschte Client-Software erstellt. Wie die Laufzeitumgebung der Software-Plattform diese Funktionalität dann auf dem jeweiligen Betriebssystem genau realisiert, bleibt ihr überlassen und ist innerhalb der Software-Plattform und für die Client-Software nicht von Relevanz. Client-Software mit Hilfe von portablen Software-Plattformen zu erstellen, ist daher eine relevante und verbreitete Alternative zu betriebssystemabhängiger Client-Software. Nachfolgend werden dementsprechend Beispiele für derartige Plattformen vorgestellt.

2.4.1 Vorteile der Portabilität

Die Vorteile von portablen Laufzeitumgebungen für Client-Software sind leicht ersichtlich. Die Portabilität wird gänzlich oder zumindest zum größten Teil von der Software-Plattform garantiert und ist damit nicht mehr Aufgabe des Programmierers der Client-Software. Das bedeutet unter anderem auch weniger Aufwand bei der Pflege des Programmcodes. Der Quelltext ist einheitlich und gültig für alle Betriebssysteme. Zudem lässt sich die Programmierung selbst oft durch Paradigmen wie die objektorientierte Programmierung besser strukturieren, aufteilen, modularisieren und damit auch warten. Außerdem können bei diesen Plattformen mitunter fehlerträchtige Programmierkonzepte wie die Zeigerarithmetik unter C[1] oder die strenge Typisierung von Variablen entfallen, die für die Funktionalität eines Programms nicht relevant sind, für die Optimierung des Programms allerdings schon.

[1] Zeiger oder *Pointer* sind in C ausgezeichnete Variablen, die eine Speicheradresse enthalten, also auf einen bestimmten Speicherbereich zeigen. Durch Dereferenzierung der Zeiger kann dann auf die dort enthaltenen Daten oder den Code zugegriffen werden.

2.4.2 Beispiele

Die bekanntesten Vertreter für portable Software-Plattformen sind derzeit die Java Standard Edition (SE), die von Microsoft initiierte .NET-Laufzeitumgebung und das offene und standardisierte Python, das vor allem im Linux-Bereich eingesetzt wird. Während Java von vornherein als portabel ausgelegt war, ist .NET zunächst eine spezifische Entwicklung für Windows-Betriebssysteme gewesen. Alle drei Software-Plattformen sind mit Einschränkungen für die gängigen Betriebssysteme Windows, OS X und Linux verfügbar. Es existiert keine offizielle Unterstützung von .NET für OS X und Linux, da es teilweise funktional abhängig von Windows-Betriebssystemen ist. Das quelloffene Projekt Mono realisiert allerdings eine Teilmenge der Funktionalität von .NET auch unter OS X und Linux durch von Microsoft veröffentlichte Spezifikationen.

Java SE

Die Java-Technologie besteht streng genommen aus drei Komponenten. Die Programmiersprache Java bietet die Formulierungsmöglichkeit für auszuführende Programme. Sie wurde unter Einfluss von Smalltalk und C++ (siehe 2.2.3 und 2.2.3) objektorientiert entworfen und ist im Jahr 1995 erstmalig erschienen. Das Entwicklungspaket Java Development Kit (JDK) enthält zusätzliche Bibliotheken und Programme zur Übersetzung von Java-Programmen in ein rechnerunabhängiges Zwischenformat, welches als Bytecode bezeichnet wird. Dieser kann von der betriebssystemabhängigen Laufzeitumgebung Java Runtime Environment (JRE) ausgeführt werden. Der Bytecode bildet zusammen mit der JVM eine Abstraktionsschicht von der Rechnerarchitektur. Die JRE führt dabei die virtuelle Maschine JVM aus, die den Java-Bytecode auf jedem Betriebssystem architekturunabhängig immer auf die gleiche Weise interpretiert. Die virtuelle Maschine muss dies durch eine korrekte Übersetzung als *betriebssystemabhängige* Software realisieren. Die JVMs sind daher nicht zwischen Betriebssystemen portabel, sondern nur die Java-Programme beziehungsweise der Java-Bytecode.

Unzählige Bibliotheken zur grafischen Darstellungen oder für Algorithmen und das Klassenkonzept mit wiederverwendbaren Programmbestandteilen begünstigen unter Java eine effiziente Entwicklungsarbeit an Programmen. Darüber hinaus bietet die Laufzeitumgebung neben der gewöhnlichen Ausführung als Desktop-Anwendung weitere unterschiedliche Schnittstellen an. Es existieren eingeschränkte und in den Webbrowser integrierte Programme, genannt Applets, vom Server für den Client ausgeführte Programme, genannt Servlets, oder Techniken zur stärkeren Bindung an die Netzwerkfunktionalität wie Java WebStart, die jedoch nicht zu den in diesem Abschnitt

betrachteten klassischen portablen Software-Plattformen gehören, daher werden sie an anderer Stelle wieder aufgegriffen.

Dem vorhandenen Overhead durch die Umwandlung in Bytecode und den damit einhergehenden zwangsläufigen Geschwindigkeitseinbußen begegnet Java mit Techniken wie der Just-in-Time-Kompilierung und speziellen architekturabhängigen Optimierungen für die JVM, die dann als Hotspot-VM bezeichnet wird. Zudem arbeitet innerhalb der JVM ein sogenannter Garbage Collector, der nicht mehr benötigte Objekte und Variablen aus dem Speicher entfernt, um die Effizienz des Programms zu erhöhen. Diese Aussagen beziehen sich allesamt auf die Referenz-Implementation der JVM der vertreibenden Firma Oracle. Daneben existieren beispielsweise für Echtzeitanforderungen oder mobile Geräte andere Implementationen der JRE und des JDK. Seit 2006 wird die Entwicklung des JDK und der JRE unter dem Namen OpenJDK weitergeführt und wurde zuvor als offener Quelltext unter der GNU General Public License (GPL) veröffentlicht [40]. Dennoch unterscheidet sich dieser Quelltext von den Versionen, die als Binärdateien heruntergeladen werden können, da nicht alle Quelltext-Bestandteile der ursprünglichen Implementation unter anderem aus Geschäftsinteressen übernommen werden konnten oder sollten.

.NET

Die .NET-Laufzeitumgebung erschien 2002 in ihrer ersten Version für Windows-Betriebssysteme und ihr Quelltext wurde 2008 in Teilen offengelegt [22]. Dies ermöglichte es dem freien Projekt Mono eine grundlegende Unterstützung von .NET-Programmen auch unter linux- und unixartigen Betriebssystemen inklusive OS X zu realisieren. Die Weiterentwicklung von Mono wird derzeit von der Firma Xamarin vorangetrieben [56]. Die Arbeitsweise von .NET ähnelt der von Java stark. Aus den Sprachdialekten der .NET-Familie (darunter C#, J# und VB), die sich an bestehende Sprachen wie C und Java anlehnen, erzeugt der .NET-Compiler den Zwischencode Common Intermediate Language (CIL). Dieser wird von der jeweiligen .NET-Laufzeitumgebung Common Language Runtime (CLR) als Teil des Betriebssystems in einer virtuellen Maschine ausgeführt. Auch .NET verwendet dazu Techniken wie Just-in-Time-Kompilierung. Die Funktion der virtuellen Maschine wird wie bei Java theoretisch durch einen Kellerautomaten realisiert [GWM01], das heißt Befehle des Zwischencodes und dessen Ergebnisse werden nacheinander auf einen einzigen virtuellen Stapel (engl. Heap) gelegt, so dass immer nur das letzte Element zugreifbar ist. Dieses Konzept erleichtert vor allem die Implementation der virtuellen Maschinen für den Zwischencode auf

verschiedenen Rechnerarchitekturen wegen der so möglichen Beschränkung auf wenige Speicherregister der CPU [GWM01].

Python

Im Unterschied zu Java und .NET existiert bei der 1991 definierten multiparadigmatischen Programmiersprache Python keine virtuelle Maschine mit generiertem Zwischencode, sondern es wird in Betriebssystemen als Skriptsprache direkt durch einen eigenen Interpreter verarbeitet [41]. Python besitzt Eigenschaften einer funktionalen, dynamischen und objektorientierten Programmiersprache. Es ist äußerst vielseitig anwendbar und es existieren aufgrund der offenen Standardisierung unzählige Bindungen an andere Programmbibliotheken und Programmiersprachen wie beispielsweise die Bindung an C-Bibliotheken des Betriebssystems, um bestimmte Programmbestandteile einer Python-Anwendung effizienter implementieren zu können. Interpreter für Python sind für alle herkömmlichen Betriebssysteme verfügbar.

2.5 Netzwerkgestützte Software-Plattformen

Dem Nachteil einer doppelten Installation der Software-Plattform und der eigentlichen Client-Software und der Wartung beider wurde durch neue Konzepte des Bezugs der Client-Software Rechnung getragen. Während die Installation der portablen Software-Plattform auf konventionelle Art und Weise nach wie vor unumgänglich ist, wird für die Client-Software durch eine bestimmte Umgebung eine netzwerkgestützte Installation und Wartung ermöglicht. Sobald der Nutzer im Webbrowser eine solche webbasierte Programmdatei öffnet und er die entsprechende Laufzeitumgebung installiert hat, übernimmt die Laufzeitumgebung der Software-Plattform die Installation beziehungsweise die Ausführung des Programms. In vielen Fällen ist dieser Schritt nur noch durch einen Klick des Nutzers zu bestätigen, wodurch der Aufwand zur Einrichtung einer Client-Software minimiert werden kann.

Die Vertrauenswürdigkeit der Software wird über eine Signatur des Herstellers geprüft und das Vertrauen hierzu beim Nutzer abgefragt. Dadurch ist es den Anwendungen möglich, dem Kontext des Webbrowsers zu entfliehen, der ohne eine entsprechende Signatur und einer Bestätigung des Nutzers die Ausführung der Anwendung auf ein Mindestmaß innerhalb des Webbrowsers beschränkt. Die Anwendungen haben so keinen Zugriff auf das Dateisystem des Rechners oder andere systemkritische Bereiche. Ist eine Signatur vorhanden und vertraut der Nutzer dieser, können die aufgerufenen

Anwendungen mehr Systemrechte erhalten und als installierte Programme im jeweiligen Bereich des Betriebssystems verankert werden. Das Konzept der Installation tritt in diesen Fällen für den Nutzer fast nicht mehr in Erscheinung, technisch gesehen wird sie aber vollzogen, indem Programmbestandteile in dafür vorgesehene Bereiche des Dateisystems kopiert werden.

2.5.1 Smart-Client-Konzept

In Analogie zu den Begriffen *Thin Client* und *Fat Client*, die eine für den Anwendungszweck bestimmte minimale beziehungsweise maximale Funktionalität beim Zugang zu Server-Diensten und -Techniken bieten, hat sich der Begriff *Smart Client* entwickelt. Smart Clients werden durch die hier vorgestellten netzwerkgestützten Software-Plattformen realisiert, so dass Installations- und Wartungsaufwand bei automatischer Aktualisierung für den Nutzer minimiert werden [Rin10].

2.5.2 Anwendungsgebiet

Die Verfügbarkeit einer ständigen Internetverbindung bietet prinzipiell auch die Möglichkeit automatischer Updates von Client-Software. Bei klassischen Desktop-Anwendungen ist das unter Umständen nicht einfach realisierbar, denn dafür wird in der Regel ein zusätzliches Programm benötigt, welches gleich einem Installer mit höheren Nutzerrechten ausgeführt wird und die eigentliche Desktop-Anwendung oder Teile davon aktualisiert. Netzwerkgestützte Software-Plattformen besitzen die Fähigkeit, die installierte Anwendung automatisch vom ausliefernden Server zu aktualisieren, falls dort eine neuere Version vorliegt und erleichtern damit die Wartung der Client-Software.

2.5.3 Beispiele

Zur Betrachtung dieser Art von Software-Plattformen und der Smart-Client-Technik werden exemplarisch die Plattformen Adobe Integrated Runtime (AIR) und Java Web Start herangezogen. Sie bilden ein Bindeglied zwischen klassisch installierbaren Desktop-Anwendungen und Web-Anwendungen im Webbrowser. Das beschriebene Smart-Client-Konzept kann auch durch die portable Software-Plattform .NET bereitgestellt werden und wird dort durch die Technologie *ClickOnce* ergänzt. Ziel dieser Technologie ist die einfache Ausführung und automatische Aktualisierung der Client-Software sowie eine minimale Benutzerinteraktion bei ihrer Installation [38]. ClickOnce wird von

Microsoft nur für windowsbasierte Betriebssysteme spezifiziert, so dass es unter Linux und OS X auch mit der Unterstützung von Mono nicht vorausgesetzt werden kann.

Adobe Integrated Runtime (AIR)

Die Firma Adobe bietet eine eingeschränkt portable netzwerkgestützte Laufzeitumgebung namens Adobe Integrated Runtime (AIR) [CDH07] an. Deren Entwicklungsbasis wird Flex genannt, welches auch zur Erstellung von Flash-Programmen in Webbrowsern verwendet werden kann. Es ist ein Konglomerat aus der an JavaScript angelehnten Skriptsprache ActionScript und einer sich an der Extensible Markup Language (XML) orientierenden Beschreibungssprache MXML. Der Funktionsumfang variiert dabei je nach Einsatzgebiet der erstellten Anwendung. So unterliegen Flash-Anwendungen im Webbrowser beispielsweise größeren Beschränkungen als AIR-Anwendungen für den Desktop, obwohl beide mit Hilfe von Flex erstellt werden. Der Fokus von AIR liegt auf netzwerkgestützten Desktop-Anwendungen, die als sogenannte *Rich Internet Applications (RIA)* außerhalb des Web-Browsers fungieren, daher können auch Hypertext Markup Language (HTML) und JavaScript bei der Erstellung einer solchen Anwendung genutzt werden. Es existierten zunächst Laufzeitumgebungen für Windows, OS X und Linux. Die Linux-Unterstützung wurde mittlerweile eingestellt und auch der virtuelle Markt für AIR-Anwendungen im Internet wurde zum Ende des Jahres 2011 geschlossen, so dass die zukünftige Nutzung der AIR Probleme bei der Verteilung und Portabilität nach sich ziehen könnte.

Java Web Start

Im Abschnitt 2.4.2 wurde bereits erwähnt, dass Java mehrere Schnittstellen für die Realisierung von Programmen bereitstellt. Während klassische Desktop-Anwendungen, die von der Java Standard Edition ausgeführt werden, normal installiert und gewartet werden müssen, verfolgt Java Web Start [Mar01] die Idee einer Verknüpfung mit dem Netzwerk. Java Web Start-Programme sind speziell gepackte Java-Programm-Archive und werden über den Webbrowser mit Hilfe des Java Network Launching Protocol (JNLP) aufgerufen. Dieses Protokoll definiert durch XML die Eigenschaften der auszuführenden Anwendung und reicht sie weiter an die JRE, die einen Web Start-Interpreter enthält. Die Web Start-Anwendung wird in einem Zwischenspeicher installiert, wenn der Nutzer das akzeptiert. Dort bleibt sie erhalten, bis eine neuere Version auf dem Web-Server verfügbar ist. Ist das der Fall, wird die neuere Version in den Zwischenspeicher geladen und diese zur Ausführung genutzt. Java Web Start-Anwendungen werden

nicht im Webbrowser-Kontext ausgeführt, besitzen allerdings zunächst dieselben eingeschränkten Rechte. Erst durch eine gültige und vom Nutzer akzeptierte Signatur der Anwendung können die Beschränkungen aufgehoben werden. Das Prinzip der Aktualisierung ähnelt der Arbeitsweise eines Webbrowsers und Java Web Start vereint es mit der Möglichkeit, dennoch eine vollwertige Desktop-Anwendung ausführen zu können.

2.6 Webbrowser als Software-Plattformen

Zur Darstellung von Text, Bildern und Links durch die textbasierte Auszeichnungssprache HTML wurden zu Anfang des World Wide Webs Programme benötigt, die über das HTTP-Protokoll eben diese Inhalte empfangen und darstellen konnten. Seit dieser Zeit hat sich der Anwendungsbereich von Webbrowsern stetig erweitert und es existieren Websites wie zum Beispiel für Online-Bank-Geschäfte oder Wikis zur Sammlung von Informationen, die dem Nutzer als Web-Anwendung eine interaktive Funktionalität bieten. Die Entwicklung in Richtung vollwertiger Software ist absehbar und technische Grenzen von Webbrowsern werden kontinuierlich verschoben, um die Funktionalität zu erhöhen.

Die Begriffe Website, Web-Anwendung und Rich Internet Application (RIA) überlappen sich teilweise, daher wird im Folgenden auf diese Begriffsklärung Bezug genommen:

- eine *Website* stellt grafische und textliche Inhalte dar und bietet eine rudimentäre Interaktion mit dem Nutzer durch Mausklicks, Tastatureingaben, sowie einfache Download- und Upload-Funktionen, sie dient in erster Linie der Präsentation von Inhalt,
- eine *Web-Anwendung* (auch Web-Application oder *Rich Internet Application*) stellt dem Nutzer einen höheren Grad an Interaktivität als eine Website zur Verfügung und erfüllt eine nachvollziehbare Funktion wie herkömmliche Anwendungen für Betriebssysteme (beispielsweise das Bearbeiten und Versenden von Mails, das Zeichnen von Grafiken oder das Schreiben von Texten).

Die Grenze zwischen beiden Begriffen ist fließend und rührt ebenso von der raschen Entwicklung der Webtechnologien und der Erschließung neuer Anwendungsbereiche her. Eine klare Trennung ist mitunter als Betrachter und Nutzer der oben genannten Inhalte gar nicht möglich, da Wissen über die technische Realisierung fehlt. Web-Anwendungen werden oft gestützt durch serverseitige Techniken, die in Abschnitt 2.1.4 erwähnt wurden, um Einschränkungen des Webbrowsers zu umgehen. Websites

und Web-Anwendungen werden in der Regel ohne eine explizite Installation direkt ausgeführt, es sei denn, der Webbrowser fordert aufgrund der empfangenen Daten eine externe Software-Plattform zur Ausführung an. Software-Plattformen außerhalb des Webbrowser-Kontextes wurden in den vorigen Abschnitten 2.2, 2.4 und 2.5 näher erläutert.

Webbrowser bilden bis zum heutigen Tage keine einheitliche Plattform für die von ihnen interpretierten Inhalte. Sie unterscheiden sich hierin und in ihren Fähigkeiten stark. Statistische Aussagen zur Verbreitung von einzelnen Webbrowsern unterliegen einerseits einer größeren regionalen Abhängigkeit und nähern sich andererseits im Mittel wie in Abbildung 2.2 zu sehen einer Gleichverteilung an. Die Portabilität von Web-Anwendungen zwischen den Webbrowsern stellt nach wie vor eine große Herausforderung dar und Kompatibilitätsprobleme von Web-Anwendungen sind nicht selten, da Webbrowser die drei Komponenten HTML, Cascading Style Sheets (CSS) und JavaScript, aus denen eine Web-Anwendung besteht, unterschiedlich interpretieren.

Zum Zeitpunkt der Erstellung dieser Arbeit sind die drei in 2.2 erwähnten Webbrowser marktüblich, die alle durch verschiedene Darstellungsroutinen, sogenannte Rendering-Engines, die grafische Erscheinung einer Website beeinflussen: Firefox, Internet Explorer und Chrome. Sie unterstützen in ihren Darstellungsroutinen jeweils unterschiedliche Befehle zur Website-Gestaltung, sowie Filter zur grafischen Manipulation und zusätzlich andere native Techniken.

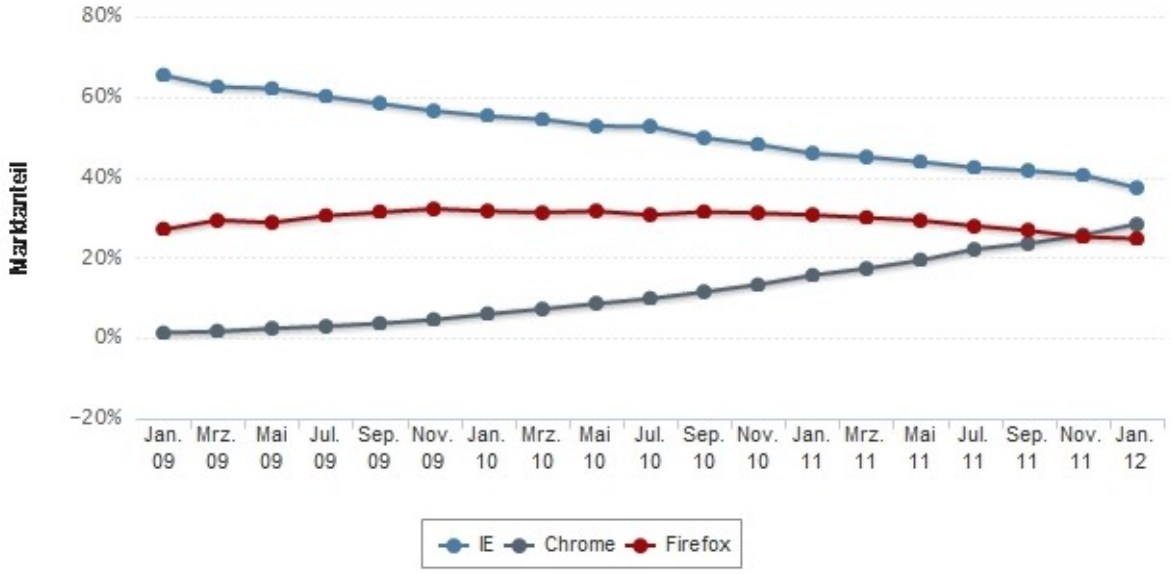

Abbildung 2.2: Webbrowser-Marktanteile an der Internetnutzung (nach [50])

Neben diesen Webbrowser-abhängigen Faktoren begünstigt der schnelle Wandel von

Standards bei den Webtechnologien die fehlerhafte Erstellung von Websites und Web-Anwendungen. Ein nicht repräsentativer Validierungs-Test staatlicher Websites der USA wurde in [32] mit dem Ergebnis durchgeführt, dass etwa 97% der Websites Fehler enthalten. Während für XML Validierungsschemata existieren, die eine Überprüfung der Korrektheit ermöglichen und Voraussetzung für die Verarbeitung sind, ist dies bei HTML, CSS und JavaScript nicht der Fall.

2.6.1 Webbrowser-Implementationen

Webbrowser sind in der Regel portable Plattformen für mehrere Betriebssysteme. Auch Plugins und Erweiterungen für die Webbrowser sind in den meisten Fällen portabel. Die hohe Zahl an Internetnutzern [26] lässt außerdem den Schluss zu, dass Webbrowser als Plattformen für Web-Anwendungen weit verbreitet sind und daher im Besonderen interessant für die Evaluation einer möglichen installationsfreien Client-Software sind, da potentiell ein großer Nutzerkreis für die zu erstellende Web-Anwendung erreicht werden kann.

Nachfolgend werden die drei nach Abbildung 2.2 am häufigsten genutzten Webbrowser vorgestellt. Es sind darüber hinaus viele andere Webbrowser wie Opera, Safari oder IceWeasel verfügbar, deren Bedeutung allerdings entweder marginal ist oder deren Technik bereits den vorgestellten Webbrowsern ähnelt, so dass die hier vorgenommene eingeschränkte Betrachtung dennoch repräsentativ ist.

Firefox

Firefox ist ein freier Webbrowser der Mozilla Foundation mit einem hohen Marktanteil, der seit 2002 entwickelt wird und von Dave Hyatt und Blake Ross initiiert wurde [53]. Die Software der Mozilla Foundation wie der Webbrowser Firefox und das Mail-Programm Thunderbird basiert auf der Programmiersprache XML User Interface Language (XUL) und der zugehörigen Laufzeitumgebung XULRunner. Mit Hilfe der Beschreibungssprache XUL können Programmoberflächen gestaltet und funktionell umgesetzt werden. Das Rendering, also die grafisch korrekte Darstellung der entstandenen Oberfläche und der Websites, übernimmt die sogenannte Gecko-Engine, welche auch in anderen Webbrowsern zum Einsatz kommt. Eine grafisch effiziente, funktionsreiche und schnelle Darstellung von Inhalten ist bei Web-Anwendungen eine Anforderung, um den Programm-Ablauf für den Nutzer interaktiv und ansprechend gestalten zu können. XULRunner muss für jedes Betriebssystem portiert werden, die Beschreibungssprache

an sich ist allerdings unabhängig davon. Massiv erweitern lässt sich die Funktionalität des Webbrowsers durch installierbare Extensions (vgl. dazu Abschnitt 2.6.4), die größtenteils als Open Source frei zur Verfügung gestellt werden.

Chrome

Der Webbrowser Chrome wird seit Ende 2008 von der Firma Google vertrieben. Er benutzt die Rendering-Engine WebKit zur grafischen Darstellung, so wie auch der unter OS X verbreitete Webbrowser Safari. Chrome ist keine Open-Source-Software, da nicht alle Komponenten als Quelltext verfügbar sind. Es existiert jedoch eine offengelegte Basis des Projekts namens Chromium, die diese Anforderungen erfüllt. Der auffälligste technische Unterschied zum Firefox besteht in seiner Behandlung der Browser-Tabs, die bei Chrome als jeweils eigener Prozess ausgeführt werden. Sollte einer davon abstürzen, wirkt sich das nicht auf die anderen Prozesse aus. Chrome-Entwickler sind zudem sehr aktiv bei der Implementation und Beschreibung neuer Techniken für Hypertext Markup Language Version 5 (HTML5), was durch die zahlreichen öffentlichen Arbeitsentwürfe bei der Web Hypertext Application Technology Working Group (WHATWG) und dem World Wide Web Consortium (W3C) nachvollzogen werden kann.

Internet Explorer

Die erste Version des Microsoft Internet Explorers erschien 1995 exklusiv für das Betriebssystem Windows 95. In den Jahren darauf wurde der Webbrowser an die Windows-Versionen gekoppelt, so dass er in diesen weit verbreiteten Betriebssystemen faktisch immer der erste installierte Webbrowser war. Ein Marktanteil von bis zu 85% war die Folge. Spezielle Anpassungen für Windows wie ActiveX und eigene definierte Skriptsprachen wie JScript führten zu einer technischen Abgrenzung von anderen Webbrowsern, was Kompatibilitätsprobleme verursachte. Die gleiche Funktionalität musste dadurch beispielsweise für mehrfach für unterschiedliche Webbrowser-Plattformen programmiert werden, wenn sie auf allen Plattformen zugänglich sein sollte. Andere Betriebssysteme wie OS X wurden zeitweilig unterstützt, in der aktuellen Version ist der Internet Explorer allerdings wiederum auf das Windows-Betriebssystem beschränkt.

2.6.2 Interpretation von Markup, Stilvorlagen und JavaScript

Die Basis für heutige Websites und Web-Anwendungen stellen die drei Techniken HTML, CSS und JavaScript dar. HTML bildet mit den Definitionen für den Dokumen-

tentyp, Meta-Informationen und der Website-Struktur die Grundlage des dargestellten Inhalts. Das sogenannte Markup durch Tags, also die Auszeichnung von bestimmten Bereichen einer Website durch Anfangs- und Endbefehle, beschreibt dabei einen gegliederten Aufbau, also beispielsweise die Kopfzeile, Fußzeile oder das Menü einer Website. Diese Bereiche können wiederum in Aussehen und Inhalt durch als CSS definierte Stilvorlagen und Funktionen in JavaScript dynamisch manipuliert werden. Die erste Version von HTML wurde 1992 spezifiziert, nachdem es am Conseil Européen pour la Recherche Nucléaire (CERN) in der Schweiz entwickelt worden war [21]. HTML in Version 4.01 ist seit 1999 ein stabiler Standard für die Erstellung von Websites und Web-Applikationen [55]. Die Extensible Hypertext Markup Language (XHTML) sollte diesen zunächst mit einer Neuformulierung von HTML innerhalb der XML-Spezifikation ablösen, wurde jedoch schließlich zugunsten von HTML5 aufgegeben, welches momentan den Status eines Arbeitsentwurfs (engl. Working Draft) hat. Die marktüblichen Webbrowser unterstützen allerdings zumindest teilweise schon die Beschreibung von Websites mittels HTML5.

Seit etwa 1996 existieren mit den CSS Stilvorlagen für die grafische Darstellung von Websites [LB99]. Die Formatierung und die Beschreibung des Aussehens wurde damit von der Strukturierung durch das Markup und dem eigentlichen Inhalt getrennt. Zusätzlich ermöglicht die einfache Syntax der CSS eine simple Anpassung oder Änderung der Darstellung wie beispielsweise den Farbwechsel des Hintergrunds oder die Änderung der Schriftgröße. Es können sehr schnell neue Stile entworfen oder Formate und grafische Änderungen vorgenommen werden, ohne in die Struktur eingreifen zu müssen. Die Definition für das Format eines bestimmten Markups (z.B. eines Absatzes) muss außerdem im Gegensatz zur Formatierung durch HTML nur einmal erfolgen. Die Fähigkeit zur Vererbung der Eigenschaften minimiert den Design-Aufwand weiterhin. Zusätzlich sind grafische Effekte wie Schattenwurf und Farbverläufe mit CSS realisierbar.

Den entscheidenden Schlussstein für eine Website bildet JavaScript. Es hat die Aufgabe, Inhalt nachzuladen und anzuzeigen oder einzelne Teile der Website dynamisch je nach Programmverlauf zu verändern. Außerdem ist es in der Lage, direkt mit dem Nutzer zu interagieren. Ein ereignisgesteuertes System innerhalb von JavaScript löst bestimmte Prozeduren aus, wenn zum Beispiel Mausklicks auftreten oder Tasten des Keyboards gedrückt werden. JavaScript ist mittlerweile auch dazu fähig, parallel zur Darstellung im Hintergrund Daten zu verarbeiten und nachzuladen. Das ist eine entscheidende Eigenschaft für interaktive Software, die maßgeblich Daten aus dem Netzwerk empfängt und auch versendet. Bei hinreichender Komplexität einer Software muss Nebenläufigkeit vor allem dann möglich sein, wenn Daten nur langsam aus dem Netzwerk bezogen

werden können. Ursprünglich blockierte JavaScript die komplette Website bis die Daten eingetroffen waren. Diese Neuerung wird heute unter dem Schlagwort Asynchronous JavaScript and XML (AJAX) zusammengefasst.

Die Bedeutung an JavaScript für installationsfreie Client-Software im Webbrowser nimmt auch derzeit noch stetig zu. Es wird kontinuierlich um Funktionen erweitert, die durch die Webbrowser-Entwickler entworfen und implementiert werden. Wie in Abschnitt 2.3 bereits erörtert, kann die Mächtigkeit der Skriptsprache damit zu einem Performance-Problem werden. Nachweisen lässt sich das über Benchmarks wie SunSpider, die die Ausführungsgeschwindigkeit von komplexen JavaScript-Programmen messen [3]. Dabei scheint der Ansatz der Just-in-Time-Kompilierung von JavaScript eine sinnvolle Lösung zu sein, wie Vergleiche mit älteren Webbrowsern zeigen, bei denen JIT-kompilierte Anwendungen um den Faktor 10 schneller abgearbeitet werden konnten [44].

HTML und CSS werden vom W3C [25, LB99] standardisiert. HTML5 ist bisher kein Standard, sondern befindet sich zum Zeitpunkt der Erstellung der Arbeit in der Entwurfsphase. Der Sprachkern von JavaScript basiert auf ECMAScript [18]. Es existieren jedoch unterschiedliche Implementierungen wie Microsofts JScript und auch Webbrowsereigene Erweiterungen der Sprache, so dass nicht jedes JavaScript von jeder Plattform korrekt interpretiert wird. Abbildung 2.3 listet noch einmal die drei kooperierenden Paradigmen der Programmierung für Web-Anwendungen und deren aktuelle Standards auf.

	HTML	*JavaScript*	*CSS*
Funktion	Website-Aufbau	Programmierung	Design, Layout
Aktuelle Standards	4.01 (1999)	1.8 (2008)	2.1 (2011)

Abbildung 2.3: Komponenten von Web-Anwendungen

2.6.3 Plugins

Nicht nur die native Interpretation der Techniken aus dem vorigen Abschnitt ist eine Möglichkeit zur Realisierung einer Web-Anwendung. Ebenso kommen funktionserweiternde Plugins für Webbrowser in Betracht, die im Folgenden einzeln vorgestellt werden. Funktional notwendig waren diese Plugins mitunter sogar aufgrund der Einschränkungen von HTML, um die Interaktivität einer Web-Anwendung zu steigern. Die Basis für die Auswahl bildet die Statistik des IT-Dienstleisters DreamingWell [16] in

Abbildung 2.4, die die verbreitetsten Plugins für Webbrowser erfasst. Der Funktionsumfang des nativ interpretierten HTML5 und den hier aufgeführten Webbrowser-Plugins konkurriert in einigen Bereichen wie zum Beispiel dem Upload von Dateien oder der beschleunigten Darstellung von 3D-Inhalten. Die Plugins können als gleichwertige oder mitunter sogar funktional höherwertige Alternative zu den nativ vom Webbrowser interpretierten Techniken gesehen werden. Allerdings erfordern sie auch eine Anpassung an das Betriebssystem und den Webbrowser. Die Anpassung erfolgt dabei oft in Form einer Installation.

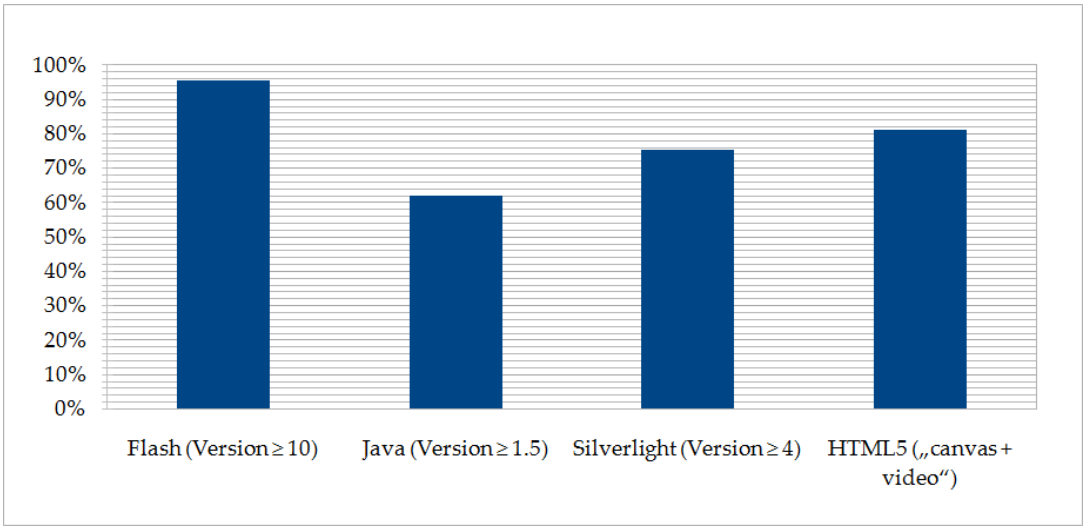

Abbildung 2.4: Verbreitung von Webbrowser-Techniken (nach [16])

Flash

Der Flash-Player gehört seit der Version 1 im Jahr 1997 zu den meist genutzten Browser-Plugins, was aktuelle Statistiken belegen [16]. Während sich Webprogrammierer seit geraumer Zeit die Fähigkeiten von Flash zunutze machen, um Animationen zu erstellen, Videos abzuspielen oder Spiele zu realisieren, bestehen gewisse Zweifel an der Zuverlässigkeit von Flash aufgrund von in der Vergangenheit aufgetretenen Anfälligkeiten. Dennoch bietet die Flash-Plattform in Anbetracht der Größe des Plugins viele Fähigkeiten für die Realisierung von Web-Anwendungen[2]. In der aktuellen Version 11 vom Oktober 2011 wurde zudem eine Stage-3D-API genannte Schnittstelle für beschleunigte 3D-Inhalte wie Spiele eingeführt, welche mit den 3D-Entwicklungen von

[2] Das aktuelle Plugin der Version 11 hat eine Größe von 18 MB, wohingegen die Java-Runtime bspw. auf einem Rechner ca. 100 MB belegt.

HTML5 namens Web Graphics Library (WebGL) konkurriert. Flash-Programme werden über ein Frontend grafisch als Filmstreifen („MovieClip“) entwickelt und mittels ActionScript programmiert. Beide Prinzipien sind einfach zu erlernen und zu benutzen, was zweifelsfrei als ein Argument für die Erfolgsgeschichte von Flash herangezogen werden kann. Die bereits vorgestellte AIR besitzt durch die gemeinsame Basis Flex viele Schnittmengen zu Flash, so dass Flash- und AIR-Anwendungen mit geringem Aufwand in das jeweils andere Format übertragen werden können. Flash ist unter anderem für die Webbrowser Internet Explorer, Firefox und Chrome verfügbar.

Silverlight

Im April 2007 veröffentlichte Microsoft erstmals ein Framework für Web-Anwendungen namens Silverlight, aktuell ist heute die Version 5. Offiziell werden nur die Windows- und OS X-Varianten der Webbrowser-Plugins zur Verfügung gestellt. Für andere Systeme existiert zumindest teilweise die von Novell vorangetriebene Implementierung *Moonlight*, welche die Funktionen von Silverlight 1 und 2 unterstützt und auf Mono basiert (vgl. 2.4.2). Die in Abbildung 2.4 ersichtliche hohe Verbreitung von Silverlight lässt sich womöglich durch die Verfügbarkeit des Plugins für alle gängigen Windows- und OS X-Webbrowser erklären, darunter Internet Explorer, Firefox, Opera und Safari.

Java Applets und Java FX

Ein weiteres Anwendungsgebiet von Java ist die Anbindung an einen Webbrowser über ein Plugin. Innerhalb des Webbrowsers dürfen jedoch nur bestimmte Java-Programme ausgeführt werden, die von der Klasse *Applet*, einem Kunstwort für für *kleine Anwendungen*, abgeleitet sind. Das Applet läuft in einem komplett vom Betriebssystem abgeschotteten Bereich nur innerhalb der JVM, das heißt es hat keinen direkten Zugriff auf die Ressourcen des Betriebssystems, sofern die JVM korrekt funktioniert. Es wird über ein spezielles HTML-Tag aufgerufen. Das Plugin ist für den Internet Explorer, Firefox und Chrome erhältlich. Applets sind ausschließlich in Bytecode ausgelieferte Programme und sie sind an die grafischen, teilweise nicht mehr als zeitgemäß betrachteten Bibliotheken der JRE gebunden. Für RIAs sind sie aus diesen beiden Gründen kaum zu gebrauchen. Java FX baut auf der Applet-Technik auf und bietet zusätzlich die Möglichkeit, Skripte zu verfassen und im Zusammenspiel mit HTML und CSS vollständige und optisch ansprechende Web-Anwendungen zu erstellen. Die Ausrichtung von Java FX ist daher Flash und Silverlight sehr ähnlich.

2.6.4 Webbrowser-abhängige Erweiterungen

Neben den bereits vorgestellten Plugins für Webbrowser, die deren Funktionsumfang erweitern, existieren auch Schnittstellen, die den Webbrowser selbst verändern können und zum Zweck haben, bestimmte Funktionen besser oder überhaupt erst zu realisieren. Diese Erweiterungen sind in jedem Fall webbrowserspezifisch und zum Teil auch betriebssystemabhängig. Sie werden in der Regel mit Hilfe der Programmierumgebung erstellt, die auch für die Erstellung des Webbrowsers genutzt wird und können somit eine hohe Funktionalität erreichen. Dadurch heben sie sich auch von den vorher beschriebenen Web-Anwendungen ab, die immer im Kontext der Einschränkungen durch den Webbrowser betrachtet werden müssen. Da es keinen einheitlichen Standard für solche Erweiterungen gibt, sind derzeit unterschiedliche Implementationsmöglichkeiten vorzufinden. Für die Webbrowser Firefox, Internet Explorer und Chrome werden diese im Folgenden kurz vorgestellt.

Firefox XPI / XUL

Erweiterungen für Firefox werden wie der Webbrowser selbst in XUL verfasst und anschließend in ein Format namens Cross-Platform Install (XPI) gepackt. Tatsächlich handelt es sich dabei lediglich um einen ZIP-Container mit einer bestimmten Ordner- und Metadatenstruktur, welcher vom Webbrowser beim Download erkannt und dem Nutzer zur Installation angeboten wird. Es existiert die Möglichkeit, die Installationsdatei kostenpflichtig signieren zu lassen, damit Firefox die Erweiterung als sicheres Programm einstuft. Größtenteils wird jedoch davon Gebrauch gemacht, die erstellten Erweiterungen durch den Internetdienst *addons.mozilla.org* überprüfen zu lassen und dort zur Verfügung zu stellen, da dies für die Entwickler kostenlos ist.

Internet Explorer ActiveX

ActiveX ist eine auf Windows beschränkte Erweiterung des dort vorhandenen Component Object Model (COM) für aktive ausführbare Inhalte. Genutzt wird diese Technik zum Beispiel für die Installation von Software über den Internet Explorer oder die Aktualisierung des Systems. Kritik daran wird vor allem deswegen geübt, weil ActiveX ohne eine webbrowserbasierte Absicherung ausgeführt wird und keine eigenen Sicherheitsmechanismen vorsieht, außer der notwendigen Bestätigung des Nutzers vor dessen Ausführung. ActiveX-Softwarekomponenten lassen sich beispielsweise in VBScript oder C++ verfassen und können direkt durch HTML eingebunden werden,

deren Ausführung der Nutzer jedoch explizit zustimmen muss. Die Laufzeitumgebung für ActiveX wird nur von Windows und seinen Bibliotheken zur Verfügung gestellt, daher ist es nicht portabel.

Chrome Extensions

Für den Browser Chrome existiert ebenfalls eine Erweiterungsschnittstelle, welche bereits vorhandene Web-Techniken wie HTML, JavaScript, JavaScript Object Notation (JSON) und CSS unterstützt und abgesehen von Chrome selbst keine spezielle Laufzeitumgebung benötigt. Die softwaretechnische Erweiterung des Browsers unterscheidet sich damit fast nicht von der Erstellung von Web-Anwendung und ist damit prinzipiell einfach gehalten. Allerdings sind die Erweiterungsmöglichkeiten im Gegensatz zu der von Firefox genutzten Programmiersprache XUL eingeschränkter, da es bei Chrome beispielsweise nicht möglich ist, eine weitere Statusleiste hinzuzufügen. Diese Funktion wird bei den Firefox-Erweiterungen häufig für Erweiterungen wie einen Nachrichtenticker verwendet.

Webbrowser-Erweiterungen sind immer abhängig vom jeweiligen Webbrowser und zudem oft noch von dessen Version. Während ActiveX in manchen Fällen sogar als sicherheitskritisch eingestuft werden kann, fehlen den Chrome Extensions wichtige Erweiterungsfunktionen. Das erfolgreichste Modell mit vielen verbreiteten Erweiterungen bietet derzeit Firefox, allerdings sind bei einem Versionssprung des Webbrowsers die Erweiterungen durch die eingetragene Version in der eigenen Meta-Daten-Struktur häufig *automatisch inkompatibel*. Insgesamt scheinen diese Techniken daher kaum geeignet, auch zwischen Webbrowsern portable und zuverlässige Web-Anwendungen zu erstellen.

2.7 Versionskontrollsysteme

Für das Beispiel einer Client-Software für eine Versionsverwaltung in dieser Arbeit ist ein kurzer Überblick der Techniken von Versionskontrollsystemen notwendig. Daher wird in diesem Abschnitt kurz auf deren Grundlagen eingegangen. Vor allem in kollaborativen Umgebungen, in denen mehrere Menschen gleichzeitig an den selben informationstechnischen Projekten arbeiten, sind Versionskontroll- oder Versionsverwaltungssysteme verbreitet. Bei der Programmierung von Software-Projekten sind sie fast immer anzutreffen, denn erst sie ermöglichen eine sinnvolle und effiziente verteilte Arbeitsweise mit Quelltexten. Versionsverwaltungen speichern alle vorgenommenen

Änderungen an Dateien und Ordnern als eigenständige Versionen (auch *Revisionen* genannt) ab und stellen diese zur Verfügung. Zudem lösen sie das Problem von verteilter Arbeit an dem gleichen Quelltext, indem sie Unterschiede zusammenführen (engl. merge) können oder zumindest auf entstandene Konflikte aufmerksam machen. Das Erkennen von Konflikten und deren Behandlung ist eine schwer zu lösende Aufgabe, deren Algorithmen Gegenstand von Untersuchungen in anderen Veröffentlichung wie in [Men02] sind.

2.7.1 Zentraler und verteilt dezentraler Ansatz

Versionsverwaltungen lassen sich grob nach ihren Eigenschaften „zentral“ und „verteilt dezentral“ klassifizieren und bieten jeweils unterschiedliche Vorteile, welche ausführlich in [54] und [AS09] diskutiert werden. Bei zentralen Versionsverwaltungen existiert nur eine maßgebliche Instanz der versionierten Daten in einem zentralen Datenspeicher, dem sogenannten *Repository*. Änderungen an den dort enthaltenen Daten werden nach dem Herunterladen lokal an Arbeitskopien vorgenommen. Die Arbeitskopien werden erst durch das Hochladen (engl. commit) in das Repository unter einer neuen Revisionsnummer eingepflegt (vgl. dazu auch Abbildung 2.5).

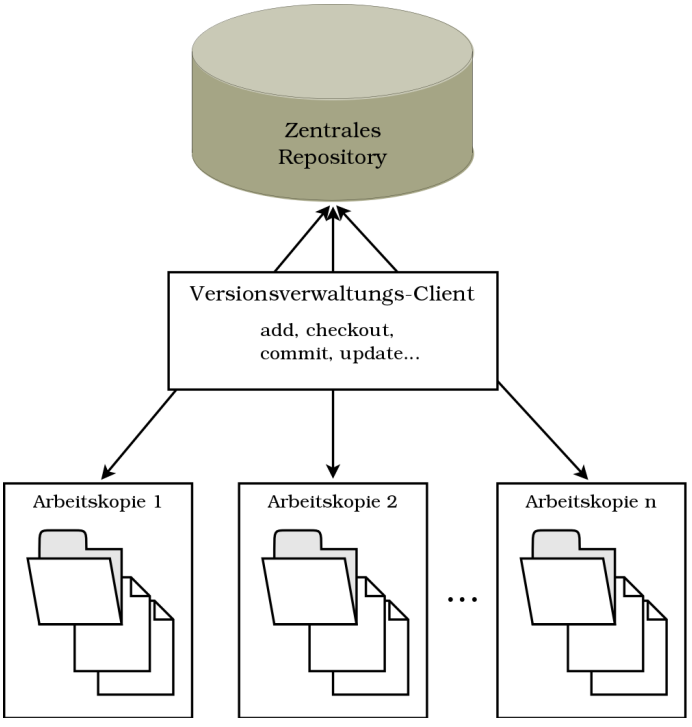

Abbildung 2.5: Arbeitsweise eines zentralen Versionsverwaltungssystems

Im Gegensatz dazu besitzen bei verteilten dezentralen Versionsverwaltungssystemen

alle Nutzer eine gleichwertige komplette Basis der versionierten Daten, auf welchen sie Änderungen durchführen. Die Synchronisation der unterschiedlichen Daten-Basen erfolgt zwischen den Nutzern mittels Übertragung von berechneten Differenzen, sogenannten Patches. Vorteilhaft an dezentralen Versionsverwaltungen ist, dass sie nicht von der Verfügbarkeit des zentralen Repositorys abhängig sind und die Verteilung der Daten bei allen Nutzern eine zusätzliche Sicherung darstellt. Dieser Ansatz wird schematisch in Abbildung 2.6 wiedergegeben.

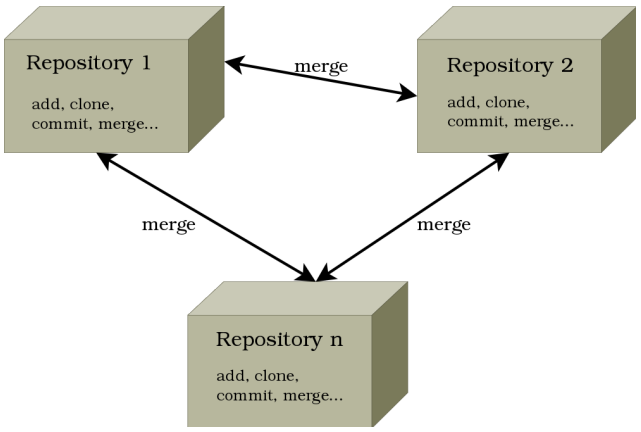

Abbildung 2.6: Arbeitsweise eines verteilten dezentralen Versionskontrollsystems

Zahlreiche Implementationen von Versionsverwaltungssystemen wie zum Beispiel Subversion, git, Mercurial, Bazaar und dem Concurrent Versions System (CVS) wurden in [Moc09] näher im Hinblick auf ihre Anwendung bei öffentlichen Quelltexten untersucht. Subversion und git sind unter anderem auch deshalb weit verbreitet, weil Anbieter von Speicherplatz für Open-Source-Software wie *sourceforge.net* und *github.com* diese Versionsverwaltungen großflächig anbieten. Subversion ist ein zentrales Versionsverwaltungssystem, git ist ein verteiltes dezentrales Versionsverwaltungssystem. Ihre Eigenschaften sollen jeweils stellvertretend für den zentralen und den verteilten Ansatz zur Versionsverwaltung sein und kurz vorgestellt werden.

2.7.2 Subversion als zentrales Versionsverwaltungssystem

Als Nachfolger des zentralen Versionsverwaltungssystems CVS wurde Subversion im Jahr 2001 mit dem Ziel veröffentlicht, die Einschränkungen von CVS zu beheben [14]. Eine Einschränkung von CVS besteht zum Beispiel darin, dass es Momentanaufnahmen (engl. Snapshots) nicht von einem kompletten Datenbaum, sondern nur dateibasiert registriert. Dadurch müssen für Revisionsänderungen die Versionsgeschichten aller

Dateien neu geschrieben werden, was bei einem großen Datenaufkommen schnell ineffizient wird. Auch Umbenennungen von Dateien führen bei CVS dazu, dass sämtliche vorhergehenden Versionen der Datei verloren gehen. Subversion löst diese Probleme und implementiert zudem die Möglichkeit die Versionsverwaltung über eine Erweiterung des HTTP-Standards zu verwenden.

Kommunikation mittels HTTP, WebDAV und DeltaV

Der in [5] definierte Standard des HTTP unterstützt eine überschaubare Menge an Funktionen, die der Client beim Server abfragen kann, welche als *Verbs* bezeichnet werden. Sendet der Client beispielsweise das Verb `GET` mit dem Parameter `index.html`, dann antwortet der Server im Normalfall mit einer erfolgreichen Statusmeldung und sendet dem Client die Datei `index.html` zu. HTTP erlaubt auf diese Weise ebenso das Senden von Daten zum Server. Es fehlen jedoch Mechanismen zur Kontrolle der Daten auf dem Server selbst. Diese wurden durch die HTTP-Erweiterung Web Distributed Authoring and Versioning (WebDAV) [19] hinzugefügt, welches auch Funktionen wie `MOVE` oder `COPY` bietet, mit denen Dateien auf dem Server bewegt oder kopiert werden können. Der Begriff *Versioning* in WebDAV führt etwas in die Irre, da der Standard keine eigentlichen Funktionen zur Versionierung von Dateien unterstützt. Diese Funktionen wurden erst durch eine abermalige Erweiterung von WebDAV um die Versionierungsfunktionen von DeltaV hinzugefügt [13]. Subversion nutzt für die Abwicklung der Versionsverwaltung über das HTTP eine Teilmenge der Funktionen von WebDAV und DeltaV. Es setzt keinen vollständigen DeltaV-Server und auch keinen vollständigen DeltaV-Client um.

Funktionsweise

Der Zugriff auf die versionierten Daten erfolgt bei Subversion nicht zwangsläufig über HTTP und seine Erweiterungen. Es ist allerdings eine sehr gebräuchliche Methode, da für den verbreiteten Standard-Web-Server namens Apache ein Modul namens `mod_dav_svn` entwickelt wurde, welches den einfachen Zugriff auf ein Subversion-Repository über den Web-Server mittels HTTP selbst ermöglicht.

Es existieren ebenso die Möglichkeiten, lokal oder über einen Socket (siehe Abschnitt 2.1.1) auf die Dateien zuzugreifen. Die Funktionsweise von Subversion ändert sich dadurch allerdings nicht. Wenn ein Client Dateien aus dem Repository laden möchte, kommuniziert er zunächst mit dem Server über die zur Verfügung stehenden Optionen.

Im einfachsten Fall erkundigt er sich nach der aktuellsten Version der Dateien und lädt sie anschließend herunter. Die lokale Speicherung erfolgt in einem Dateibaum mit der gleichen Struktur der Daten im Repository. Auf diese Weise entsteht auf dem Client-Rechner eine Arbeitskopie der Daten, an welcher Änderungen vorgenommen werden können. Ist die Änderung abgeschlossen und veranlasst der Nutzer das Hochladen, berechnet der Client die entstandenen Änderungen über einen Differenzen generierenden Algorithmus. Diese Differenzen macht der Client beim Server bekannt, der daraus eine neue Revision erzeugt. Subversion überträgt Daten standardmäßig in komprimierter Form, um die Kommunikationseffizienz zu erhöhen. Die Prozesse des Herunterladens und des Hochladens werden bei Versionsverwaltungen in der Regel als *Checkout* bzw. *Commit* bezeichnet.

2.7.3 git als verteiltes dezentrales Versionsverwaltungssystem

git wurde ursprünglich als Versionsverwaltungssystem für die Weiterentwicklung des Linux-Kernels entworfen. Es unterscheidet sich stark von der Arbeitsweise des zentralen Versionsverwaltungssystems Subversion, weshalb es hier Erwähnung findet. In git besitzt jeder Nutzer das komplette Projektarchiv inklusive der Versionsgeschichte. Es gibt daher kein zentrales Repository, sondern nur gleichberechtigte Projektarchive, die mit Hilfe von für das effiziente Zusammenfügen implementierte Algorithmen synchronisiert werden können [12]. Revisionen werden in git mittels eines kompletten Snapshots des Projektarchivs und einer kryptografischen Hash-Funktion erzeugt, wodurch unter anderem Fälschungssicherheit für die Entstehungsgeschichte genau dieser Version hergestellt werden kann. Ein Vorteil von git ist, dass sämtliche Änderungen lokal in dem eigenen Snapshot des Projektarchivs vorgenommen werden. Es besteht erst wieder die Notwendigkeit einer Netzwerkverbindung, wenn die Änderungen mit anderen geteilt werden sollen. Der Austausch von Daten zwischen git-Repositories kann über mehrere Verbindungsprotokolle erfolgen, unter anderem SSH und HTTP.

2.7.4 Konfliktbehandlungsmodelle

Der Vorteil von Versionsverwaltungen, vielen Nutzern gleichzeitig Zugang zu Daten und deren Speicherung zu verschaffen, stellt in bestimmten Situationen einen ebenso großen Nachteil dar. Versuchen zwei Nutzer simultan auf dieselbe Datei Änderungen in der Versionsverwaltung anzuwenden oder können die jeweiligen Arbeitskopien nicht zusammengefügt werden, entsteht ein Konflikt, der behoben werden muss. Für die

Lösung solcher Probleme gibt es zwei gebräuchliche Modelle, die auch in Subversion und git Anwendung finden [14].

Sperren – Ändern – Entsperren

Das Sperren-Ändern-Entsperren-Modell linearisiert die Änderungen an einer Datei im Repository, indem es sie dort exklusiv für einen aktuellen Nutzer reserviert, der sie ändern darf. Gleichzeitig wird die Datei für die anderen Nutzer aber gesperrt und erst nach erfolgter Änderung wieder entsperrt. Hierbei können jedoch neue Probleme auftreten, zum Beispiel wenn der Nutzer vergisst, die Datei zu entsperren. Auch wenn zwei Nutzer parallel an verschiedenen aber untereinander abhängigen Dateien Änderungen vornehmen, die hinterher nicht mehr zueinander kompatibel sind, entsteht wiederum ein Konflikt. Dieses Modell ist bei verteilten Versionskontrollsystemen wie git nicht vorhanden, da jedem Nutzer eine eigene komplette Kopie der versionierten Daten vorliegt und kein zentrales Repository existiert. Eine zentrale Sperrung ist daher gar nicht möglich.

Kopieren – Ändern – Zusammenfassen

Ein anderer Ansatz wird beim Kopieren-Ändern-Zusammenfassen-Modell verfolgt. Hier arbeitet jeder Nutzer auf eigenen Kopien der vorhandenen Daten des Repositorys. Am Schluss werden die Änderungen aller Nutzer zu einer neuen Version zusammengefasst und diese ins Repository eingepflegt. Natürlich ist auch das Zusammenfassen hier nicht frei von Konflikten. In manchen Fällen können Menschen diese Konflikte manuell beheben, in anderen Fällen ist ein Zusammenfassen von vornherein nicht möglich wie zum Beispiel oft bei Binärdateien.

3 Evaluation und Eigenschaften installationsfreier Client-Software

Nach der eingehenden Betrachtung des aktuellen Stands der Technik und der Vorstellung von Software-Plattformen im vorigen Kapitel, beschäftigt sich das folgende Kapitel zunächst mit der Evaluation dieser Techniken im Hinblick auf installationsfreie Client-Software. Im Besonderen wird auf den Begriff *installationsfrei* eingegangen und erarbeitet, was diesen auszeichnet. Dazu werden zunächst allgemeine Software-Eigenschaften erörtert, um eine Basis für die nähere Untersuchung von installationsfreier Client-Software herzustellen. Die Eigenschaften von installationsfreier Client-Software werden in diesem Zusammenhang schließlich genauer bestimmt. Am Ende des Kapitels werden die Anforderungen des konkreten Beispiels einer Versionsverwaltung dahingehend untersucht, ob und inwiefern eine technische Umsetzung mittels installationsfreier Client-Software möglich ist.

3.1 Aspekte von Software im Allgemeinen

Software und Hardware sind keine gegensätzlichen Begriffe. Prinzipiell kann eines das andere ersetzen und Software kann als *immaterielle Hardware* angesehen werden [MS00]. Während Hardware allerdings nach ihrer Erstellung in den meisten Fällen nicht mehr geändert werden kann, ist es möglich, Software und damit Programme für bestehende Hardware weiterzuentwickeln. Welche Funktionen in Software und welche in Hardware umgesetzt werden, ändert sich im Laufe der Zeit [MS00], daher stellt die Untersuchung von Software in dieser Arbeit eine Momentaufnahme aktueller Techniken dar und erhebt keinen Anspruch auf Vollständigkeit aller Möglichkeiten zur Lösung einer softwaregestützten Aufgabe.

3.1.1 Software als Mittel der Informationstechnologie

Ein System der Informationstechnologie (IT) transportiert Datenströme aus Bits, der kleinsten informationstechnischen Dateneinheit, durch Zeit und Raum [MS00]. Die

Kommunikation zwischen Client und Server ist dabei die räumliche Übertragung der Datenströme, während ein Speicher diese zeitlich aufbewahrt. Die Datenverarbeitung durch einen Prozessor ändert die Datenströme an einem bestimmten Punkt der Raumzeit. Nach [MS00] ist Software einerseits immateriell wie *Information*, andererseits ist sie nur sinnvoll, wenn sie tatsächlich etwas *vollbringt*. Sie gleicht mit diesen Eigenschaften am ehesten einem Dienst in einer Dienstleistungsgesellschaft, denn auch Software benötigt eine Instanz, besser gesagt einen Prozessor, der die durch sie angestrebte Intention verwirklicht. Software manifestiert sich in der Regel in einem Programm, das Befehle ausführt, um damit etwas Nützliches zu bezwecken, daher werden beide Begriffe teilweise synonym verwendet. Die Informationstechnologie beziehungsweise die Informatik im wissenschaftlichen Bereich beschäftigen sich mit der Bearbeitung, Speicherung und Übertragung von Informationen in Form von Bildern, Text und anderen Datenformaten. Software ist in der Lage, diese Informationen mit Hilfe von Mustererkennung sinnvoll zu verarbeiten [MS00].

3.1.2 Eigenschaften

Software erfordert eine Umgebung in der sie ausgeführt werden kann. Isolierte Software ist sinnlos und kann keine Funktion erfüllen [MS00]. Zu dieser Umgebung gehören je nach Anforderungen der Software verschiedene Kriterien, einige davon sind jedoch fast immer notwendig. Im Folgenden werden daher kurz wesentliche und allgemeine Eigenschaften bei der Ausführung von Software aufgeführt und erläutert. Zweck dieser genaueren vorausgehenden Erörterung ist die Bestimmung und Diskussion der Eigenschaften von installationsfreier Client-Software, die bis dato auch bezogen auf ihre Eigenschaften nach Kenntnis des Autors nicht eindeutig definiert sind.

Rechenzeit

Zum Wesen einer Software gehört, dass sie Rechenzeit des Prozessors des informationstechnischen Systems benötigt, um ihre Funktion zu erfüllen. In welchem Umfang diese benötigt wird und zur Verfügung steht, hängt von der jeweiligen Aufgabe und dem Computersystem ab. In modernen Betriebssystemen wird Rechenzeit für Software durch einen Zeitplaner (sog. Scheduler) bereitgestellt, der der Software bestimmte Anteile an der Rechenkapazität des Prozessors zeitgesteuert zuweist [SG98]. Im einfachsten Fall werden alle Programme reihum für eine bestimmte Zeitspanne mit Rechenzeit versorgt und müssen dann wieder pausieren.

Haupt- oder Primärspeicher

Die zweite wesentliche Komponente von auszuführender Software ist ein für die Software verfügbarer Hauptspeicher, auch Primärspeicher genannt [SG98]. Ein Programm ist nur in der Lage, Befehle abzuarbeiten, wenn es diese Befehle lesen und zu verarbeitende Daten lesen und schreiben kann. Beschrieben wird dieser Aufbau beispielsweise durch die *von Neumann-Architektur* [Neu93] oder die *Harvard-Architektur* [Wil56]. Software wird heutzutage in der Regel von einem nichtflüchtigen Speicher in den Hauptspeicher des Rechners geladen und ausgeführt [SG98]. Sie kann dort auf eigene Speicherbereiche zugreifen, um ihre Befehle und Algorithmen auszuführen. Der Primärspeicher ist ein flüchtiger Speicher. Die dort gespeicherte Software und ihre angelegten Daten werden nach der Ausführung zum Überschreiben freigegeben [SG98].

Nichtflüchtiger Sekundärspeicher

Zusätzlich zum flüchtigen Arbeitsspeicher existiert Sekundärspeicher [SG98], der Daten physikalisch fest ablegen kann. Nichtflüchtiger Speicher unterscheidet sich von flüchtigem Speicher durch die Fähigkeit, Daten dauerhaft halten zu können, auch wenn er nicht mit Energie versorgt wird. Dies geschieht oft magnetisch auf einer Festplatte oder über optisch erkennbare eingeprägte Muster auf Kunststoffdatenträgern wie bei CDs und DVDs. Für die meiste Software ist nichtflüchtiger Speicher unerlässlich. Das Betriebssystem muss beispielsweise von einem derartigen Speicher geladen werden und jedes E-Mail-Programm wäre sinnlos, wenn die E-Mails nicht auch nach dem Wegfall der Energieversorgung gespeichert blieben. Technisch gesehen greift Software dabei nicht direkt auf den Festspeicher zu, sondern über eine logische Abstraktion in Form von Dateien und Ordnern, die vom Dateisystem des Betriebssystems zur Verfügung gestellt wird. Das Betriebssystem nutzt seinerseits Gerätetreiber zum Austausch von Daten mit Sekundärspeichergeräten wie Festplatten oder CD-Laufwerken [SG98].

Netzwerk- und Kommunikationsfähigkeit

Software muss in vielen Fällen mit anderer Software kommunizieren können, um ihre Funktion zu erfüllen. Die Kommunikation beinhaltet dabei den Austausch von Daten zwischen zwei Teilnehmern, die unterschiedlicher Art sein können. So existiert beispielsweise die Interprozesskommunikation zwischen zwei nebenläufigen Teilprozessen desselben Programms [SG98] oder die Kommunikation zwischen Client- und Server-Software über Sockets innerhalb eines Netzwerks. Ersteres ermöglicht eine schnellere,

weil parallele und gleichzeitige Abarbeitung von Befehlen und wird als Multi-Threading bezeichnet, wobei ein Thread einen Teilprozess darstellt [SG98]. Eine mögliche Form der Kommunikation für Letzteres wurde in Abschnitt 2.1.1 genauer beschrieben. Unzählige Protokolle können zur Datenübertragung genutzt werden. Sie unterscheiden sich zum Beispiel darin, ob sie einen Übertragungsstatus aufrechterhalten oder in welcher Form die übertragenen Daten vorliegen.

Nutzung von lokalen Hardwareressourcen

Der direkte Zugriff einer Software auf Hardware-Ressourcen ist größtenteils nicht notwendig und auch nicht möglich. Der Zugriff wird in der Regel vom Betriebssystem zum Beispiel durch abstrahierte Gerätetreiber implizit zur Verfügung gestellt [SG98]. Es gibt allerdings auch hier Ausnahmen, wenn Effizienz und Geschwindigkeit im Vordergrund stehen. Bei der Erstellung von 3D-Anwendungen können Entwickler beispielsweise auf eine programmiersprachen- und plattformunabhängige Programmierschnittstelle von Grafikkarten namens OpenGL zurückgreifen, die vereinfacht gesagt, die Ausführung von speziellen Grafik-Befehlen direkt auf der Grafikkarte ermöglicht. Die Verwendung von CPU-gestützten Software-Routinen wäre hingegen um ein Vielfaches langsamer und ineffizienter.

Besondere Privilegien bei der Ausführung

Die Ausführung eines Programms erfolgt innerhalb der gängigen Betriebssysteme auf Basis einer Identifikation als *Nutzer*. Dabei können Betriebssysteme Nutzer aufweisen, die beispielsweise nur die Aufgabe haben, ein spezielles Programm im Hintergrund auszuführen oder Wartungsarbeiten zeitgesteuert durchzuführen, wohingegen andere Nutzer eine grafische Benutzeroberfläche direkt bedienen. In Betriebssystemen wird klassisch zwischen Nutzer- und System-Modus unterschieden [SG98]. Die meisten Prozesse können im unprivilegierten Nutzer-Modus ausgeführt werden und benötigen keine besonderen Rechte. Wird allerdings von einem Prozess ein Systemaufruf wie der direkte Zugriff auf Hardware getätigt, muss der Nutzer die Berechtigung dafür besitzen. Die Berechtigung erlangt er beispielsweise durch eine gesonderte Authentifizierung oder er hat den Prozess bereits als privilegierter Nutzer gestartet. Unter Windows und Linux sind gebräuchliche Namen für diese Nutzer *Administrator* beziehungsweise *root*.

3.2 Kriterien für installationsfreie Client-Software

Die zuvor beschriebenen allgemeinen Eigenschaften von Software besitzen auch im Spezialfall von installationsfreier Client-Software Gültigkeit. Im Rahmen von allgemeiner Software ist es zunächst möglich, schon erwähnte triviale Eigenschaften auch für diese Art von Software zu fordern. Rechenzeit und Hauptspeicher sind notwendige Bedingungen, ohne die keine Software ausgeführt werden kann (vgl. auch 3.1.2). Die explizite Nutzung von lokalen Hardware-Ressourcen oder besondere Privilegien zur Ausführung lassen sich als weitere Spezialfälle zunächst einmal ausschließen, da dies eine zusätzliche Forderung nach einem privilegierten Betriebssystemnutzer aufstellen würde. Im Zusammenhang mit einer installationsfreien Anwendung von Software wäre dies aber bezüglich der Systemsicherheit bedenklich. Nichtflüchtiger Speicher ist eine Voraussetzung, um eine Software auch nach Wegfall der Energieversorgung wieder sinnvoll nutzen zu können.

Zusätzlich wird für die hier betrachtete Art von Software die Installationsfreiheit und ihre Fähigkeit der clientseitigen Netzwerkkommunikation gefordert. Außerdem ist es sinnvoll, einen gewissen Grad an Interaktivität durch eine grafische Benutzeroberfläche zu fordern, da installationsfreie Client-Software, die im Hintergrund ohne Nutzerinteraktion und damit ohne das Wissen des Nutzers ausgeführt wird, einerseits wiederum sicherheitskritisch sein kann und andererseits keinen sinnvollen Daten- und Befehlsaustausch mit dem Nutzer zulässt, der bei einer Versionsverwaltung notwendig ist.

3.2.1 Installationsfreiheit

Die Installationsfreiheit ist keine typische Eigenschaft von Software, da Software in der Regel von einem Datenträger auf einem anderen kopiert und dabei explizit für das Zielsystem eingerichtet wird [8]. Die Nutzung des Netzwerks als Software-Lieferant hat diese Methodik in gewisser Weise durch die Vorteile in der Dynamik der Verteilung und Aktualisierung von Software in einigen Anwendungsbereichen überholt, so dass die Quelle kein physikalischer Datenträger mehr sein muss. Für installationsfreie Client-Software muss in jedem Fall gelten, dass sie keine wie in [8] beschriebene, ihrer Ausführung vorausgehende, Installation benötigt. Das bedeutet im Einzelnen:

1. Es findet kein Kopieren der Software von einem Datenträger in einen persistenten Speicherbereich des Dateisystems des Betriebssystems des Zielrechners statt (der Zwischenspeicher des Webbrowsers entspricht gerade nicht dieser Definition, da er kein herkömmlicher persistenter Speicher ist),

2. es wird kein Installationsprogramm (Installer) ausgeführt, das Anpassungen jeglicher Art am Betriebssystem oder an den Einstellungen des Nutzers zur Einrichtung der Software vornimmt und
3. es sind keine besonderen Nutzerrechte notwendig für den Prozess der Initialisierung und den Start der Software.

Der Aspekt der Installationsfreiheit verlangt also gerade, dass die Software selbst *zu jeder Zeit* im Netzwerk verfügbar sein muss. Ihr Vorhandensein im Zwischenspeicher des Webbrowsers des Clients wird prinzipiell weder erwartet noch vorausgesetzt. Im Gegensatz dazu ist für den Nutzer eine persistente Speichermöglichkeit von Daten bezogen auf die Anwendung in der Regel unabdingbar.

3.2.2 Netzwerkkommunikation

Client-Software in einem Netzwerk ist ohne die Fähigkeit der Kommunikation isoliert und wiederum wenig sinnvoll, daher muss installationsfreie Client-Software zwingend in der Lage sein, mit einem Server kommunizieren zu können. Client und Server teilen sich ein gemeinsames Netzwerk als Medium, einigen sich auf ein Übertragungsprotokoll und tauschen Daten aus, die beidseitig verstanden werden können. Nach [MS00] lassen sich die Kommunikationspartner in diesem Fall als interoperabel und komplementär bezeichnen, das heißt sie sind physikalisch in der Lage, miteinander zu operieren und außerdem dazu fähig, die gegenseitig ausgetauschten Daten zu interpretieren.

3.2.3 Nichtflüchtiger Speicher

Nichtflüchtiger Speicher ist für installationsfreie Client-Software vor allem eine sinnvolle Anforderung aus Nutzersicht, da die meisten Rechnersysteme heutzutage solchen Speicher enthalten und Daten somit auch nach Wegfall der Energieversorgung aufrechterhalten können. Dies ist notwendig, um einen bestimmten Status eines Programms oder dessen Ergebnisses vorhalten zu können, auch wenn der Computer nicht mit Energie versorgt wird. Für installationsfreie Client-Software wird daher sinnvollerweise eine Form von persistentem Sekundärspeicher gefordert. Zwar sind andere Prinzipien wie eine vollständige Speicherung auch der persönlichen Daten auf einem Server denkbar, sie sind allerdings momentan nicht besonders praxisrelevant. Die technische Entwicklung des Netzwerks mit den vorhandenen Problemen wie unkontrollierbaren Verbindungsabbrüchen und hohen Latenzen scheint zur Bewältigung solcher Dienste noch nicht ausgereift genug zu sein.

3.2.4 Interaktivität

Die Interaktivität einer Software muss in der Regel durch Anforderungen an das Produkt *Software* beschrieben werden. Eingeschränkt wird dies allerdings durch die technischen Möglichkeiten einer Software oder Software-Plattform. Unter Interaktivität soll daher im Folgenden die Fähigkeit der Software verstanden werden, dem Nutzer der Software die gewünschte Funktionalität sowie die Ein- und Ausgabe von Daten auf einfache und bekannte Art und Weise zur Verfügung zu stellen. Das wird für Nutzer ohne tiefgreifende informationstechnische Kenntnisse nur durch eine simple Interaktion mit der Software möglich sein. Denkbar sind hier Sprachsteuerungen oder eine grafische Benutzeroberfläche. Ersteres ist bisher zumindest im Netzwerk kaum verbreitet und erschwert auch einen Datenaustausch, daher ist eine grafische Benutzeroberfläche zum Zeitpunkt der Erstellung der Arbeit eine sinnvolle Forderung, um mit einem Dienst wie einer Versionsverwaltung arbeiten zu können.

3.3 Vergleich von Software-Plattformen

Nach einer Begriffsdefinition für installationsfreie Client-Software im Kapitel 2, der Vorstellung verbreiteter Techniken und den im vorigen Abschnitt aufgestellten Kriterien für installationsfreie Client-Software ist nun eine Evaluation der technischen Plattformen für Software möglich. Im Mittelpunkt wird dabei die Installationsfreiheit stehen. Die Evaluation soll zudem klären, inwiefern installationsfreie Client-Software als bestimmter Typus von Software systematischen Beschränkungen unterliegt und welche Eigenschaften sich daraus ergeben. Zunächst werden *weiche* Faktoren wie die Akzeptanz und Verbreitung betrachtet, die entscheidend sein können, falls sich die technische Realisierung mit unterschiedlichen Software-Plattformen als gleichwertig erweist.

3.3.1 Akzeptanz und Wahrnehmung

Akzeptanz und Wahrnehmung einer Software sind schwer zu definierende Begriffe, da sie nutzerabhängig sind und einer repräsentativen Studie bedürfen. Aspekte positiver Wahrnehmung wie die schnelle Ausführung des Programms, ein geringer Wartungsaufwand und wenige sichtbare oder wenige durch Informationsaustausch kolportierte Implementationsfehler erhöhen die Akzeptanz einer Client-Software plausiblerweise

maßgeblich. Das Warten auf das Ende des Startvorgangs oder bestimmte Abarbeitungsroutinen der Software ist wahrscheinlich ein Hauptkriterium für die Akzeptanz einer Software. Betriebssystemabhängige Software kann beispielsweise durch Bindung an die vorhandenen Bibliotheken, die sich bereits oft im Arbeitsspeicher befinden und ihr Vorliegen in für das System kompilierter Form im Normalfall sehr schnell zur Ausführung gebracht werden.

Bei portablen Software-Plattformen wie Java und .NET steht zwischen dem Aufruf des Programms und seiner Ausführung oft zunächst einmal die zeitaufwändige Initialisierung der jeweils notwendigen Ausführungsumgebung wie der JVM oder der CLR. Zusätzlich führt der durch die Übersetzung von abstraktem Zwischencode in Maschinencode entstehende Overhead zwangsläufig zu Geschwindigkeitseinbußen gegenüber einer nativen Anwendung des Betriebssystems. Dieses Problem wurde durch die vorgestellten JIT-Compiler (siehe 2.3.2), die im Fall von Java den Bytecode und im Fall von .NET die CIL zur Laufzeit in wiederverwendbaren Maschinencode übersetzen in seiner Wirkung zumindest abgemildert.

Skriptsprachen wie die der Kommando-Shell verursachen durch die notwendige Interpretation der Befehle im Regelfall die schlechtesten Laufzeiten. Sie starten aber mitunter schneller als portable Software-Plattformen, da die in Maschinencode vorliegenden Interpreter entweder sehr klein sind oder sich bereits im Arbeitsspeicher befinden, daher können sie bei wenig umfangreichen Programmen Vorteile bei der Start- und Ausführungszeit haben.

Im Gegensatz zu den drei zuvor genannten Software-Techniken sind Start- und Laufzeitverzögerungen bei den vom Webbrowser ausgeführten Web-Anwendungen in hohem Maße und in Größenordnungen vorhanden, die die Akzeptanz und Wahrnehmung negativ beeinflussen können. Bei der Kommunikation in einem Netzwerk sind durch die physikalischen Beschränkung der Laufzeit von Signalen immer Latenzen gegeben, die sich bei zunehmender Entfernung von Client und Server noch verstärken. Führt der Nutzer die Web-Anwendung zum ersten Mal aus, muss sie zunächst komplett heruntergeladen werden. Das ist oft bereits zeitkritisch und verlangt eine Optimierung des Vorgangs durch Reduktion der übertragenen Datenmengen. Beispielsweise können Grafikformate im Internet bereits schon unscharf dargestellt werden, wenn erst ein Bruchteil der gesamten Datei vorhanden ist.

Web-Anwendungen lösen das Problem des Zeitaufwands durch die Übertragung heutzutage zum Beispiel, indem der Webbrowser auf den eigenen Zwischenspeicher zurückgreift oder der Nutzer über Fortschrittsanzeigen davon in Kenntnis gesetzt wird, dass Daten nachgeladen werden. Dies ist im Prinzip erst mit der Einführung von

nebenläufigen Datenabfragen durch AJAX möglich geworden. Webbrowser und Web-Anwendungen versuchen überdies durch Heuristiken schon vor dem Klick des Nutzers möglicherweise notwendige Daten herunterzuladen, damit die Darstellung beschleunigt wird. Darüber hinaus üben große Netzwerke wie das Internet selbst einen Einfluss auf die in ihnen verfügbaren Anwendungen aus, denn Web-Entwickler versuchen in der Regel möglichst viele Anwender zu erreichen. Dazu müssen sie Akzeptanz und Benutzbarkeit steigern. Eine Rückmeldung für den Erfolg liefern dann Statistiken über die Anzahl der Anwender beziehungsweise Besucher einer Website beziehungsweise Web-Anwendung.

3.3.2 Verbreitung

Die Verbreitung einzelner Plattformen und Technologien ist nach Ansicht des Autors kaum mit stichhaltigen Daten zu belegen, da die Erstellung repräsentativer Statistiken systematisch schwierig ist. In den meisten Fällen müsste ein spezielles Programm dazu in dem jeweiligen Betriebssystem relevante Informationen sammeln und diese weiterleiten. Eine Realisierung dessen ist aber auch aufgrund des persönlichen und geschäftlichen Interesses an Datenschutz unwahrscheinlich. Der Webbrowser sendet zwar beim Besuch einer Unique Resource Identifier (URI) ein Identifikationsmuster über seine Version und damit potentielle Fähigkeiten namens *User-Agent* an den Server, welches zur Auswertung herangezogen werden kann, dieser ist allerdings vom Nutzer beliebig änderbar und zum Teil auch nicht aussagekräftig. Nach einer Statistik von [4] über im Internet verwendete Betriebssysteme, die durch ein Browser-Plugin oder durch Identifikationen des Webbrowsers freiwillig erstellt werden kann, lassen sich jedoch zumindest einige grundlegende Aussagen treffen. In Abbildung 3.1 ist zu erkennen, dass Windows-Betriebssysteme nach wie vor stark vor OS X für Apple-Computer und Betriebssystemen für Smartphones dominieren.

Betriebssystemabhängige Client-Software muss prinzipiell für alle Betriebssysteme einzeln mit den jeweiligen Programm-Bibliotheken und Compilern übersetzt werden und wird damit in der Verbreitung gehemmt. Der durch die parallele Unterstützung von mehreren Betriebssystemen entstehende Aufwand ist sehr groß und verursacht mitunter schlecht les- und wartbare Quelltexte, in denen bei C-ähnlichen Programmiersprachen Präprozessor-Befehle den Quelltext nach Betriebssystemen gliedern. So wird vor dem eigentlichen Übersetzen des Programms lediglich der Quelltext für das aktuelle Betriebssystem herausgefiltert und inkompatibler Quelltext ignoriert. Je mehr Betriebssysteme marktüblich sind und unterstützt werden sollen, desto schwieriger wird die Erstellung einer gemeinsamen Basis der Client-Software. Mitunter ist eine

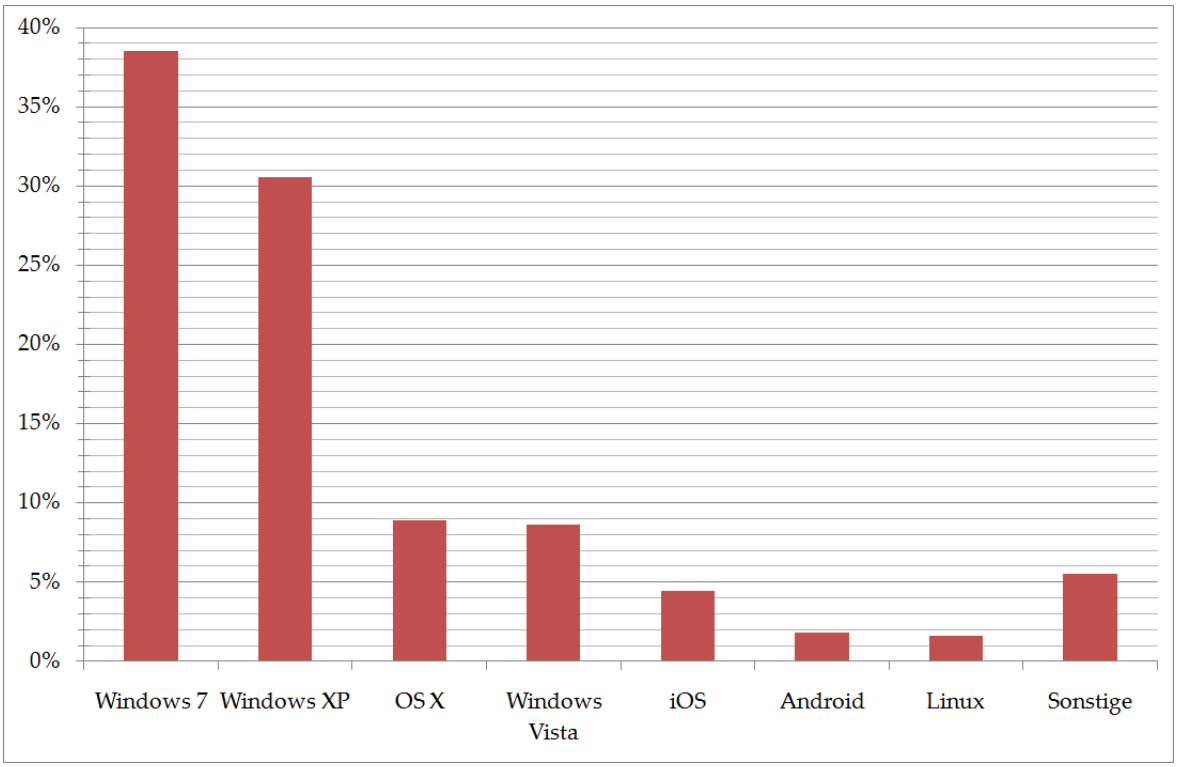

Abbildung 3.1: Verbreitung von Betriebssystemen (nach [4])

gemeinsame Basis auch gar nicht möglich, wenn die Bibliotheken und Compiler unter anderen Betriebssystemen gar nicht existieren. Portable Programmierplattformen wie Qt, die auf die vorhandene C/C++-Umgebung des Betriebssystems aufsetzen, haben sich aus diesem Grund entwickelt und auch eine gewisse Verbreitung erlangt. Insgesamt lässt sich jedoch aufgrund der Vielzahl der Betriebssysteme, Bibliotheken und Compiler keine exakte Häufigkeit einer bestimmten Variante dieser zusammenhängenden Komponenten ausmachen.

Portable Software-Plattformen sind ebenso inhomogen verteilt. Windows 7 und Windows Vista enthalten ab ihrer Installation bereits die Software-Plattform .NET in den Versionen 3.5 beziehungsweise 3.0 [39], Windows XP und OS X hingegen unterstützen diese Plattform nicht nativ. Installieren lässt sich die aktuelle .NET-Plattform in Version 4 in fast allen Windows-Versionen nach der eigentlichen Betriebssystem-Installation. OS X wird im Gegensatz zu Windows standardmäßig mit einer Laufzeitumgebung für Java ausgeliefert und die Linux-Betriebssysteme installieren in der Regel das OpenJDK. Die Verbreitung von Java kann mit Hilfe der Abbildung 2.4 geschätzt werden, da bei einem verfügbaren Webbrowser-Plugin auch die JRE installiert sein muss. Java ist daher derzeit auf ca. 60% der Rechner als portable Software-Plattform verfügbar.

Die Verbreitung von netzwerkgestützten Software-Plattformen aus Abschnitt 2.5 ist

kaum einzuschätzen. Nach der teilweisen Einstellung von AIR, bleibt zumindest Java Web Start eine verbreitete Alternative, da eine Standard-Installation der JRE wie das Webbrowser-Plugin auch immer die Unterstützung für Java Web Start installiert, so dass Java Web Start ebenfalls auf ca. 60% aller Rechner verwendet werden kann.

Webbrowser sind in Betriebssystemen unter Umständen bereits automatisch enthalten wie der Internet Explorer unter Windows oder Safari unter OS X. Es gehören allerdings weder Firefox noch Chrome zu den vorinstallierten Webbrowsern unter den verbreiteten Betriebssystemen, so dass die beiden Webbrowser mit der höchsten Innovationsdichte im Bereich HTML5 und JavaScript nicht generell vorausgesetzt werden können. Eine Alternative ergibt sich hier durch das Betriebssystem *Chrome OS*. Das ist eine gewöhnliche Linux-Distribution ist, die vorrangig auf die Nutzung von Web-Anwendungen abzielt und daher Chrome als Standard-Webbrowser oder gar als Standard-Interpreter enthält, was eine zuverlässige Entwicklung für dieses eine spezielle Betriebssystem und seine Software-Plattform Chrome erlaubt. Ansonsten existieren regional unterschiedliche Präferenzen im Bezug auf den verwendeten Webbrowser. In den USA ist der Internet Explorer nach wie vor der verbreitetste Webbrowser, in Mitteleuropa ist es der Firefox und in Südamerika überwiegend Chrome [10]. Die Verteilung von Webbrowsern ist demnach ebenfalls als sehr heterogen zu bezeichnen. Insgesamt scheinen Webbrowser allerdings die verbreitetste Software-Plattform zu sein.

3.3.3 Installationsprozess der Software-Plattform

Wird die zu erstellende Client-Software mit Hilfe einer abstrahierenden Software-Plattform wie in Abschnitt 2.3 beschrieben realisiert, muss die Software-Plattform zusätzlich auf dem Client-Rechner installiert werden. Dieser Schritt entfällt bei nativer betriebssystemabhängiger Client-Software, da die Software-Plattform mit dem Betriebssystem identisch ist.

Portable und netzwerkgestützte Software-Plattformen sowie Webbrowser müssen in der Regel als betriebssystemabhängige Software installiert werden, um eine Laufzeitumgebung für die eigentliche Client-Software bilden zu können. Plattformen wie Java, .NET (bzw. Mono unter Linux) und Python bieten dazu Installationspakete für jede unterstützte Betriebssystemversion an, ebenso verhält es sich bei AIR, Java Web Start (bzw. Java SE) und den Webbrowsern. Für den Nutzer der eigentlichen Client-Software bedeutet der Installationsschritt einer dafür notwendigen Software-Plattform zusätzlichen Aufwand. Eine Aktualisierung oder Wartung dieser findet in der Regel nicht automatisch statt, so dass es dem Nutzer obliegt, sich darum zu kümmern.

Außerdem erscheint die zusätzliche Installation einer Software-Plattform für den Nutzer erst sinnvoll, wenn mehrere Anwendungen damit genutzt werden können oder sich die Ausführung der Anwendungen dadurch vereinfacht. In vielen Fällen ist Letzteres jedoch nicht zutreffend, da sich viele auf Java- und .NET basierende Programme ebenfalls auf herkömmliche Weise installieren und dann zusätzlich die jeweilige Laufzeitumgebung benötigen.

Einen Ausweg bieten hier Kombinationen aus betriebssystemabhängigen Portable Apps und portablen Software-Plattformen, so dass der Nutzer eine speziell in dem eigenen Betriebssystem ausführbare Archiv-Datei herunterlädt und lediglich ausführt, wobei diese die portable Software-Plattform inklusive der eigentlichen Anwendung enthält. Das Entpacken und weiterleiten an die Ausführungsschicht der Software-Plattform übernimmt dann die Archiv-Datei. Im Fall von SimMoLib wird dieser Mechanismus für den auf Java basierenden Web-Client derzeit eingesetzt. Das bedeutet, der Web-Client wird verteilt inklusive einer vollständige JRE für Windows und der eigentlichen Client-Software für SimMoLib. Der größte Nachteil dieser innerhalb von Windows portablen und für den Nutzer einfachen Lösung liegt in der Größe des gesamten Programmpakets.

Aufgrund der inhomogenen Verteilung von portablen und netzwerkgestützten Software-Plattformen und Webbrowsern können für den allgemeinen Fall keine expliziten Voraussetzungen getroffen werden und es ist davon auszugehen, dass die jeweilige Plattform eine zusätzliche Installation erfordert. Im Besonderen kann das durch Einschränkungen behoben werden, indem der Betrieb der Client-Software auf ein einzelnes Betriebssystem reduziert wird, da zum Beispiel .NET unter Windows Vista und 7 oder Java unter OS X nativ vorhanden sind.

3.3.4 Installationsprozess der Anwendung

Die Vielzahl von Server-Diensten macht es unmöglich, für alle Dienste eine entsprechende native Client-Software in das Betriebssystem einzubinden. Zur Nutzung eines Server-Dienstes ist daher oft die Installation einer neuen Client-Software notwendig, die durch verschiedene Software-Plattformen oder betriebssystemabhängig realisiert werden kann. Der Vorgang unterscheidet sich mitunter im konkreten Ablauf etwas, aber in der Regel handelt es sich bei der Installationsdatei um ein Archiv, das die eigentliche Client-Software sowie Meta-Daten dazu enthält. Die Meta-Daten liefern zusätzliche Informationen über Abhängigkeiten, Versionen oder zur Vertrauenswürdigkeit des Installationspakets.

Für Windows und OS X kommen öfter eigens erstellte Installer zum Einsatz, die selbstausführende Dateiarchive sind. Unter OS X ist neben der Installer-Methode auch noch eine andere Methode der Installation gebräuchlich. Dabei wird ein komprimiertes Disk-Image einer Anwendung eingebunden. Unter unixähnlichen Betriebssystemen wie OS X und Linux wird dieser Vorgang auch als *mount* bezeichnet. Der Nutzer kann die Anwendung anschließend auf seine Festplatte kopieren. Installer hingegen haben die Aufgabe, die Installation der Client-Software in das Betriebssystem selbst korrekt durchzuführen und eventuell auch deren Entfernung wieder zu ermöglichen. Die Ausführung eines Installers verlangt in der Regel administrative Privilegien innerhalb des Betriebssystems. Problematisch ist das fehlende Wissen des Nutzers über die Art und den Inhalt des auszuführenden Installers, der auch Schadcode enthalten könnte. Das wirft Fragen nach seiner Vertrauenswürdigkeit auf, für die es derzeit keine Lösungen gibt, die vollständig sicher sind. Zertifizierungen der Installationsdateien werden unter Windows [34] mit Hilfe von Zertifizierungsanbietern vorgenommen, was ein Vertrauen des Nutzers gegenüber diesen Anbietern voraussetzt. In den meisten Fällen entscheidet der Nutzer aber mangels genauerer Kenntnisse der Technik und des Anbieters daher lediglich durch einen Mausklick über das Vertrauen und die anschließende Ausführung. Ganz ähnlich verhält es sich mit den Konzepten der Installation von Anwendungen auf Smartphones und Tablets. Die bei diesen Geräten verbreiteten Betriebssysteme iOS und Android stellen dem Nutzer einen App-Store zur Verfügung, der Anwendungen aus dem Netzwerk auf das Gerät laden kann. Die dauerhafte Installation erfolgt nach einer Bestätigung des Nutzers. Da eine Installation mit privilegierten Nutzerrechten eine Gefahr für die Systemintegrität darstellen kann, existieren verschiedene, nur teilweise eingesetzte Begrenzungsmechanismen der Manipulationsmöglichkeiten bei einer Installation. Erweiterte Systemeigenschaften[1] können beispielsweise die Auswirkungen dieser Problematik zumindest abschwächen.

Abhängigkeiten einer Anwendung müssen bei ihrer Installation aufgelöst werden. Benötigt eine Anwendung beispielsweise Java oder .NET als installierte Software-Plattform, kann der Installer das überprüfen und fordert den Nutzer gegebenenfalls dazu auf, die Plattform zuerst zu installieren. Anwendungen, die als betriebssystemabhängige Software oder mittels einer portablen Software-Plattform erstellt wurden, werden heutzutage in den meisten Fällen klassisch mit Hilfe eines Installers auf dem Betriebssystem eingerichtet. Dieser Schritt ist nicht zwangsläufig notwendig, wie die Existenz von portablen Anwendungen zeigt. Die Kapselung in einem Installer erzeugt vor allem einen gewissen Komfort für den Nutzer, indem er durch die Installation leitet und

[1] SELinux (Security Enhanced Linux) bietet zum Beispiel zusätzliche Eigenschaften für Dateien und Ordner, so dass diese auch von privilegierten Nutzern nicht einfach gelöscht werden können.

in dem Betriebssystem außerdem eine Schnittstelle zur Entfernung der Anwendung einrichtet. Darüber hinaus werden beispielsweise Umgebungsvariablen gesetzt, die die Anwendung benötigt oder es werden Seriennummern in das Betriebssystem für eine käuflich erworbene Software eingetragen usw.

Unter Linux bestehen die signierten Installationspakete [42] für Paketverwaltungen aus komprimierten Datenblöcken, welche nach dem Herunterladen entpackt werden. Die darin enthaltene Software wird in das Dateisystem kopiert. Ein Speicher für Meta-Daten der Installationspakete ermöglicht ein einfaches Entfernen der Dateien. Die Signatur des Pakets dient der Sicherstellung, dass keine Änderungen an dem Paket vorgenommen wurden. Paketverwaltungen sind allerdings nicht zwingend für ein Linux-Betriebssystem. Die Installation von Software kann auch durch das eigenhändige Kompilieren von Quelltexten in einer Art standardisiertem Verfahren vorgenommen werden. Dabei enthalten die Quelltext-Archive oft ein Konfigurations-Shell-Script namens `configure`, welches benötigte Abhängigkeiten auflöst und ein `Makefile` erstellt. Dieses wiederum enthält Übersetzungsanweisungen für den Compiler, etwa welche Rechner-Optimierungen oder Installationspfade genutzt werden sollen. Mit dem Programm `make` und Parametern wie *all* oder *install* wird schließlich die Übersetzung beziehungsweise Installation durchgeführt.

Bei Anwendungen, die mittels netzwerkgestützten Software-Plattformen realisiert wurden, vereinfacht sich der Installationsschritt bereits merklich. Java Web Start erfordert zur Einrichtung lediglich die Bestätigung des Vertrauens des Nutzers gegenüber der vorhandenen Signatur der Anwendung, nachdem diese im Webbrowser angeklickt wurde. Danach ist sie bereits in einem speziellen Programmbereich der Plattform installiert und startfähig. Eine ähnliche Funktion bietet auch AIR.

Web-Anwendungen benötigen hingegen überhaupt keine Installation. Sobald die Web-Anwendung vom Webbrowser vollständig in den Zwischenspeicher heruntergeladen wurde, wird sie interpretiert, angezeigt und ausgeführt. Web-Anwendungen entsprechen damit als einzige Anwendungen der strikten Forderung der Installationsfreiheit, auch wenn die zuvor erwähnten durch netzwerkgestützte Software-Plattformen erstellten Anwendungen dem recht nahekommen. Beide Plattform-Arten unterscheiden sich in der Initialisierung visuell nur durch eine zusätzliche Bestätigung des Nutzers. Technisch gesehen wird für die jeweilige Anwendung ein anderer Speicherbereich genutzt, aber die gleiche Aktualisierungsmethodik vom Server zum Client angewandt.

3.3.5 Portabilität

Die explizite Bindung an ein Betriebssystem schränkt die Portabilität von betriebssystemabhängiger Software massiv ein. In vielen Fällen sind die dafür genutzten Installer nicht einmal unter verschiedenen Versionen des gleichen Betriebssystems kompatibel, so dass sie oft für jede Kombination aus Rechnerarchitektur (wie 32-Bit oder 64-Bit), Betriebssystemtyp (Windows, OS X, Linux) und Betriebssystemversion (Windows XP, Windows Vista, usw.) extra erstellt werden müssen. Da dies einen enormen Aufwand verursacht, ist es üblich, Software nur für bestimmte verbreitete explizite Betriebssysteme zu vertreiben. Die bereits erwähnten Portable Apps sind zwar ohne Installation lauffähig und damit prinzipiell zwischen gleichen Betriebssystemen auf verschiedenen Rechnern portabel, jedoch in der Regel nicht zwischen verschiedenen Architekturen, Typen oder Versionen von Betriebssystemen.

Die Portabilität von Plattformen für Anwendungen, also portablen und netzwerkgestützten Software-Plattformen, sowie Webbrowsern und deren Plugins, ist in der Regel größer, da der Entwickler der Plattform dieses Ziel verfolgt. Abbildung 3.2 unterstreicht diese These, denn es ist ersichtlich, dass die Plattformen oft für mehrere Betriebssysteme verfügbar sind. Die ausgeführte Anwendung profitiert von der Portabilität der Software-Plattform, wird dadurch aber auch von einer Installation der Software-Plattform abhängig. Manche Techniken sind nur teilweise verfügbar wie etwa Mono als .NET-Plattform unter Linux und OS X, welches nicht das gesamte Funktionsspektrum von .NET abdecken kann, auch wenn Microsoft selbst die Entwicklung von Mono begünstigt hat [56]. Dennoch implementiert Microsoft in .NET und damit auch Silverlight nach wie vor Techniken, die das sogenannte Win32-Application Programming Interface (API) nutzen und damit in der Regel technisch nicht für andere Betriebssysteme portierbar sind.

Auffallend ist die fehlende Unterstützung von Java unter den mobilen Betriebssystemen Android und iOS. Android-Anwendungen werden zwar in Java geschrieben, sie werden allerdings nach dem Übersetzen in Java-Bytecode noch einmal von einem Cross-Assembler in Bytecode für die ins Android-Betriebssystem integrierte Dalvik-VM übersetzt [7] [Nic09]. Die Dalvik-VM ähnelt der JVM zwar, beide unterscheiden sich allerdings fundamental in der Abarbeitung der Befehle. Während die JVM nach dem Prinzip des Kellerautomaten arbeitet, führt die Dalvik-VM ihre Programme als Registermaschine aus, was vor allem deswegen vorteilhaft ist, weil heutige Prozessorarchitekturen ebenfalls als Registermaschinen aufgebaut sind und die Dalvik-VM in dem Zusammenhang bis zu 50% effizienter arbeitet [7].

Die höchste Portabilität wird innerhalb der in dieser Arbeit vorgestellten technischen

	Windows	*OS X*	*Linux*	*Android*	*iOS*
Java SE	✔	✔	✔	–	–
.NET / Mono	✔	•	•	•	•
Java Web Start	✔	✔	✔	–	–
AIR	✔	✔	–	–	–
Internet Explorer	✔	–	–	–	–
Firefox	✔	✔	✔	✔	–
Chrome	✔	✔	✔	•	–
HTML, JavaScript, CSS	✔	✔	✔	✔	✔
Flash	✔	✔	✔	✔	–
Silverlight / Moonlight	✔	•	•	–	–
Java FX	✔	✔	✔	–	–

Legende: ✔ *unterstützt* • *teilweise unterstützt* – *nicht unterstützt*

Abbildung 3.2: Portabilität von Software-Plattformen

Möglichkeiten derzeit mit Web-Anwendungen erreicht (vgl. Abbildung 3.2). Webbrowser und ihre Plugins wie Flash sind auf nahezu allen Rechnersystemen verfügbar, die dem Nutzer eine grafische Oberfläche zur Verfügung stellen und die im Netzwerk Dienste in Anspruch nehmen können. Unterstützt wird die Portabilität von Web-Anwendungen darüber hinaus durch sogenannte JavaScript-Frameworks wie jQuery [52], die eine einheitliche webbrowser-übergreifende Programmierschnittstelle für JavaScript bieten. Damit entfallen in der Regel notwendige Test-Szenarien für unterschiedlichste Webbrowser-Modelle und -Versionen, die alle unterschiedliche JavaScript-Standards und Abweichungen davon implementieren.

3.3.6 Aktualität der Software

Die Frage nach der Aktualität einer Software stellt sich im Zusammenhang mit der rapiden Entwicklung von Software-Plattformen vor allem bei Web-Anwendungen. Die sich im Zustand stetiger Veränderung befindlichen Technologien HTML5, JavaScript und CSS des W3C und der WHATWG erfordern einen hohen Aufwand bei der Evaluation von Software. Eine Anforderungsanalyse deckt eventuell Mängel auf, die nach kurzer Zeit durch neue Entwicklungen bereits behoben sind. Auch die Wartung nach der Erstellung einer Web-Anwendung ist problematisch, da die Webbrowser ständig aktualisiert werden müssen, um entstehenden Sicherheitsproblemen standhalten zu

können. Ähnliches gilt im Übrigen ebenso für servergestützte Web-Anwendungen wie in Abschnitt 2.1.4 erwähnt. Die Webbrowser Firefox und Chrome begegnen der rasanten Entwicklung mit automatischen Updates, die im Hintergrund ablaufen und den Nutzer lediglich kurz über die Aktualisierung informieren. Diese Technik stößt allerdings dort an Grenzen, wo der Nutzer nicht die benötigten Rechte zur Aktualisierung hat, ein Plugin für den Webbrowser veraltet ist oder der Webbrowser durch eine Paketverwaltung installiert wird, die im Folgenden keine Aktualisierung vornimmt. Die für diese Webbrowser verantwortlichen Firmen Mozilla Foundation und Google bringen andererseits regelmäßig Vorschläge für neue W3C-Standards und Erweiterungen oder Anpassungen bestehender Standards oder implementieren diese sogar schon vor einer offiziellen Spezifikation, so dass eine Evaluation mangels Dokumentation schwierig werden kann.

Auch bei den anderen aktiv genutzten portablen Software-Plattformen wie Java und .NET scheint durch die Anbindung an das Internet und Techniken wie Smart-Clients der Innovationszyklus verkürzt. Seit Mitte 2011 wird die Version 7 von Java vertrieben und für die für Mitte 2013 vorgesehene Version 8 ist unter anderem eine Interoperabilität mit JavaScript angekündigt [46]. Von .NET existiert bereits eine Vorabversion 4.5 [37]. Es ist daher offenbar sinnvoll, Strategien zur Aktualisierung von Software-Plattformen zu entwickeln und deren Eigenschaften genauer zu verfolgen.

3.3.7 Interaktivität und technische Möglichkeiten

Wesentlich ist für eine vom Menschen zu nutzende Software, welche Interaktivität und welchen Funktionsumfang sie dem Nutzer bereitstellen kann. Die bisher vorgestellten Techniken sind alle dazu fähig, eine grafisch interaktive Oberfläche zu gestalten und darüber hinaus eine Funktionalität zu erfüllen. Der Grad an realisierbarer Funktionalität ist allerdings zwischen den Plattformen und Möglichkeiten für Client-Software nicht identisch. Das liegt zum Einen an systematischen Beschränkungen wie zum Beispiel bei im Webbrowser-Kontext ausgeführten Anwendungen und zum Anderen existieren bei portablen Software-Plattformen keine Programmierschnittstellen für spezielle Funktionen des Betriebssystems wie der schon genannten Win32-API, weil diese gerade nicht portabel umsetzbar sind. Verschiedene Aspekte reduzieren somit die Interaktivität und die technischen Möglichkeiten einer Software, was in diesem Abschnitt veranschaulicht werden soll.

Alle Software-Plattformen bieten hinsichtlich ihrer Funktionalität gleichzeitig Vorteile und Nachteile. Sie sind einerseits durch die Installation unterschiedlich fest an ein

Betriebssystem gebunden, andererseits ermöglichen sie eine Installationsfreiheit, falls nur eine geringe Interaktivität mit dem Betriebssystem und dem Nutzer zur Funktionserfüllung notwendig ist. Abbildung 3.3 visualisiert diesen Zusammenhang.

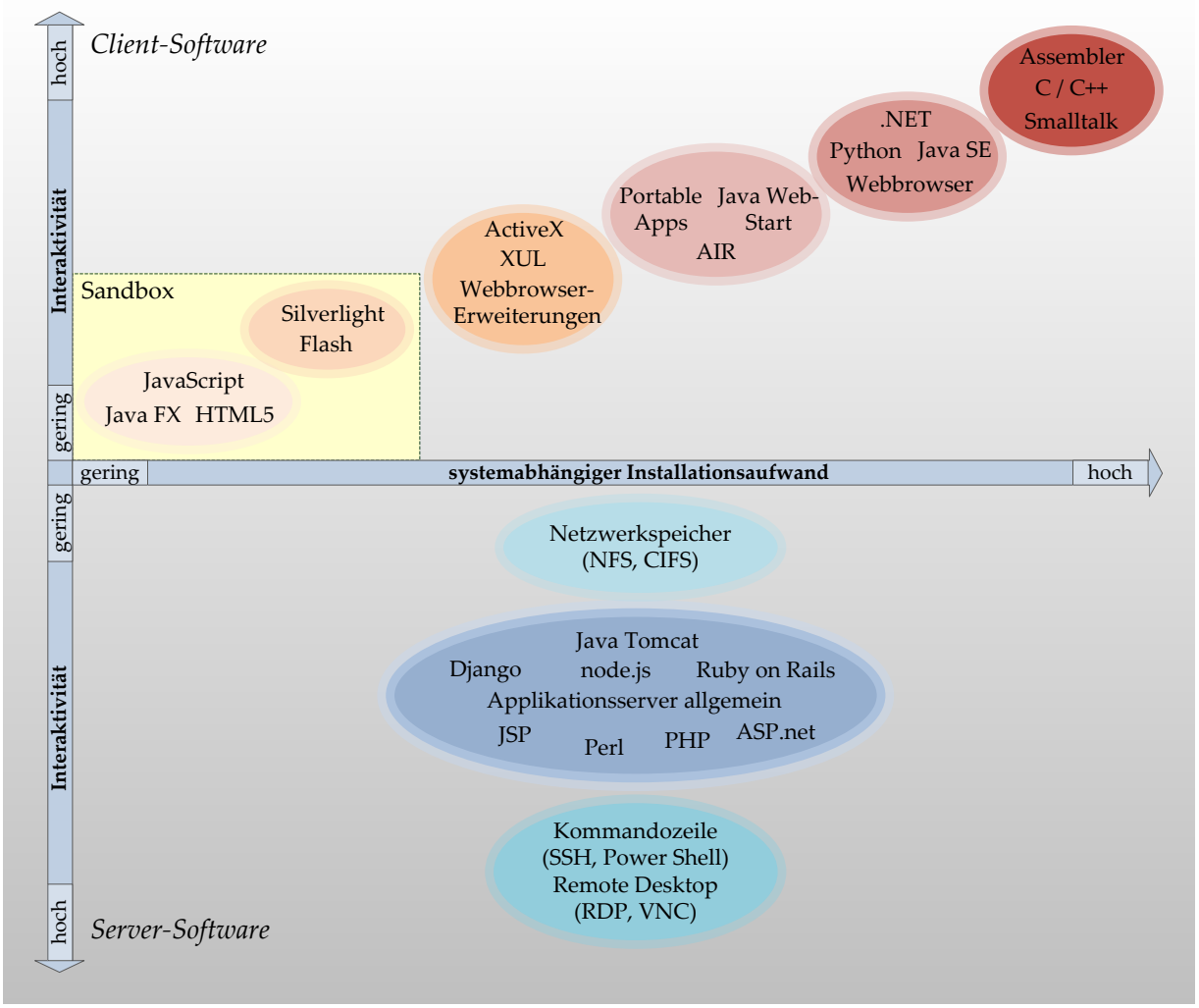

Abbildung 3.3: Interaktivität von Client- und Server-Software

Im unteren Bereich ist eine Auswahl von möglichen Techniken für Server-Software dargestellt. Diese wird der Vollständigkeit halber mit aufgetragen, da eine Client-Server-Struktur immer aus zumindest zwei unterschiedlichen Endpunkten besteht, die einerseits Software für einen Server und andererseits Software für einen Client benötigen. Server-Software lässt sich vereinfacht mit Hilfe des Begriffs der Interaktivität gruppieren. Während Netzwerkspeicher-Dienste wie NFS beispielsweise nur die Speicherung von Dateien und Ordnern erlauben, sind Applikationsserver in der Lage, mehr Funktionalität durch eine vielseitige Anwendung zur Verfügung zu stellen. Applikationsserver stehen in engem Zusammenhang mit den im Abschnitt 2.1.4 erläuterten Techniken. Ein

Server-Dienst wie *Remote Desktop*, der eine vollständige Kontrolle über den Server ermöglicht, bildet in der Grafik das Höchstmaß an Interaktivität. Der Installationsaufwand dieser Techniken variiert und ist nicht klar abzugrenzen.

Im oberen Bereich von Abbildung 3.3 sind Techniken für Client-Software im Allgemeinen dargestellt. Es ist eine Linearität zwischen der Interaktivität, also den funktionalen Fähigkeiten der Client-Software, und dem systemabhängigen Installationsaufwand zu erkennen. Diese ergibt sich aus der Betrachtung in den Abschnitten 3.3.3 und 3.3.4. Betriebssystemabhängige durch C/C++ usw. realisierte Software ist in der Lage, sämtliche Funktionen des Betriebssystems zu nutzen und kann sogar auf lokale Hardware-Ressourcen direkt zugreifen. Sie bietet damit die höchste Interaktivität, allerdings in der Regel auch den höchsten Installationsaufwand. Das liegt zum Einen an der expliziten Anpassung für die jeweilige Kombination aus dem Betriebssystem, der Betriebssystemversion und den Programm-Bibliotheken und zum Anderen an der notwendigen Integration in das Betriebssystem selbst. Unter Windows werden oft Einträge in der System-Registrierung, einer Art Speicher für Meta-Daten des Betriebssystems, oder die zusätzliche Installation eines *Uninstallers* vorgenommen. Manche Software wie Microsoft Office oder Adobe Acrobat installieren darüber hinaus sogar die Installationspakete auf dem Rechner, um im Notfall Reparaturen zu ermöglichen.

Portable Software-Plattformen wie Java, Python und auch Webbrowser selbst verzichten hingegen in den meisten Fällen auf betriebssystemabhängige Funktionen auf Kosten einer geringeren Interaktivität und verringerter technischer Möglichkeiten. Die Portabilität der Plattformen steht hierbei im Vordergrund. Die Nutzung und der Ausbau von Windows-eigenen Systemaufrufen lässt aber im Falle von .NET den Schluss zu, dass Microsoft vor allem Portabilität zwischen den Windows-Versionen vorsieht und nicht betriebssystemübergreifend, das heißt .NET könnte unter Windows prinzipiell dazu in der Lage sein, eine technisch ähnlich hohe Interaktivität zu erreichen wie betriebssystemabhängige Software.

Die netzwerkgestützten Software-Plattformen Java Web Start, AIR und andere bilden zusammen mit ActiveX, XUL und Webbrowser-Erweiterungen eine Übergangsschicht. Damit ist gemeint, dass sie oft in bestimmter Weise mit Web-Anwendungen auf der einen Seite und (portablen) Software-Plattformen auf der anderen Seite verknüpft sind. In einer Web-Anwendung ruft der Nutzer beispielsweise ein Java-Web-Start-Programm auf, welches durch das JNLP-Protokoll initialisiert wird und schließlich durch die JRE ausgeführt wird. Ebenso verhält es sich mit ActiveX, das aus der Web-Anwendung heraus windowseigene Skriptsprachen nutzen kann. Auch die Programmierung von Firefox-Erweiterungen in XUL erlaubt diesen Schritt der Funktionserweiterung. Die

Interaktivität einer Anwendung ist hier zum großen Teil von der konkreten Realisierung abhängig, denn es werden gleichwohl Absicherungsmaßnahmen für die Software wie auch zusätzliche Signatur-Verfahren für Programm-Code eingesetzt, wodurch eine höhere Interaktivität ermöglicht werden kann. Diese Techniken sind jedoch teilweise nicht portabel, so dass zusätzlich eine Betriebssystemabhängigkeit auftreten kann.

Im Abschnitt 3.3.4 wurde die Installationsfreiheit bereits als Eigenschaft von Web-Anwendungen erörtert. Technisch begrenzt wird diese Form der installationsfreien sofortigen Ausführung allerdings durch das auch in Abbildung 3.3 visualisierte Sandbox-Prinzip. Web-Anwendungen wird der Zugriff auf kritische Bereiche des Betriebs- oder Dateisystems verwehrt und eine Interaktion mit dem Nutzer ist lediglich in engen Grenzen möglich. Für viele Dienste ist das bereits ausreichend. Es existieren jedoch Anwendungsgebiete, wo eine höhere Interaktivität erforderlich ist wie zum Beispiel, wenn Daten im Dateisystem abgelegt oder eine spezielle Netzwerkverbindung hergestellt werden soll. Dazu sind Web-Anwendungen im Normalfall nicht fähig. Techniken wie JavaScript, JavaFX, HTML, Silverlight, Flash und andere bieten daher prinzipiell die geringste Interaktivität.

3.3.8 Evaluationsergebnisse

Am Anfang des zweiten Kapitels wurde definiert, welche Voraussetzungen installationsfreie Client-Software erfüllen muss, um als solche bezeichnet werden zu können. Es wird ersichtlich, dass lediglich Web-Anwendungen der Anforderung genügen, *installationsfrei* zu sein. Der Begriff *installationsfrei* lässt sich dabei mit der Arbeitsweise eines Webbrowsers sinnvoll verknüpfen. Der Webbrowser lädt die Client-Software zwar herunter, was einer klassischen Installation ähnelt, sie residiert danach jedoch in dem nutzerbezogenen Zwischenspeicher. Transparent für den Nutzer wird sie dort vom Webbrowser aktualisiert, falls der Server neue Versionen der Dateien zur Verfügung stellt. Prinzipiell ist der Zwischenspeicher jedoch ein flüchtiger Speicher, so dass die Software jederzeit auf dem Server verfügbar sein muss. Das ist ein entscheidender Unterschied zur herkömmlichen Installation, bei welcher Software im persistenten Dateisystem gespeichert wird. Die Methodik wird sowohl von portablen Software-Plattformen wie von betriebssystemabhängiger Software gewählt, daher können diese nicht als installationsfrei bezeichnet werden. Netzwerkgestützte Software-Plattformen sind *fast* installationsfrei, da der lokale Speicher für die Anwendung auch aktualisiert wird wie der Zwischenspeicher des Webbrowsers. Allerdings liegt ebendieser Speicher auch im Dateisystem des Betriebssystems, womit faktisch eine Installation vorliegt.

Zur sinnvollen Nutzung der Web-Anwendung als Client-Software muss der Nutzer nach dem Bezug dauerhaft online sein. Es existieren allerdings auch Techniken wie Google Gears, die es Web-Anwendungen ermöglichen, nach dem Bezug paradoxerweise nicht mehr „im Web“ abzulaufen, sondern offline auf den Webbrowser selbst beschränkt.

Als Plattformen und Laufzeitumgebungen für installationsfreie Client-Software dienen demnach lediglich Webbrowser wie der Internet Explorer, Firefox oder Chrome, die zuvor als betriebssystemabhängige Software-Plattform installiert werden müssen. Im Bezug auf die in dieser Arbeit zu untersuchende Client-Software erweitern wir daher nach der Klärung des Begriffs *installationsfrei* die Definition von installationsfreier Client-Software um Folgendes:

- eine *installationsfreie Client-Software* ist eine im Webbrowser-Kontext ausgeführte Web-Anwendung, die notwendigerweise einen Client für einen Server-Dienst darstellt, der durch einen Socket definiert wird.

Das Ergebnis der Evaluation zeigt, dass lediglich im Webbrowser ausgeführte Web-Anwendungen für installationsfreie Client-Software in Frage kommen. Gleichzeitig ergibt sich aus dem Prinzip der funktionalen Einschränkung durch die Sandbox eine Problematik, die einer näheren Betrachtung der technischen Möglichkeiten innerhalb der Sandbox bedarf. Diese Betrachtung wird im folgenden Abschnitt vorgenommen, der damit zugleich die Eigenschaften solcher Web-Anwendungen thematisiert.

3.4 Installationsfreie Client-Software und die Sandbox

Installationsfreie Client-Software im Webbrowser führt zwangsläufig zur Frage nach der Integrität der so ausgeführten Programme und der Sicherheit des Betriebssystems, in welchem der Webbrowser gestartet wurde. Endpunkte in der Netzwerkkommunikation können zwar über bestimmte Techniken wie das verschlüsselte Übertragungsprotokoll Hypertext Transfer Protocol Secure (HTTPS) als vertrauenswürdig verifiziert werden, das ist bei HTTP aber standardmäßig nicht der Fall. Um die Auswirkungen von Web-Anwendungen trotzdem kontrollieren zu können, wurde das Prinzip der *Sandbox* entwickelt.

Eine Sandbox (zu deutsch *Sandkasten*) kapselt eine Laufzeitumgebung für ein Programm von einer anderen zu schützenden Laufzeitumgebung ab. Sinnvoll ist dies bei Programmen wie Web-Anwendungen, deren Ursprung zwar prinzipiell durch den URI nachvollzogen, deren Vertrauenswürdigkeit aber nicht ohne Weiteres geklärt werden

kann. Die Kapselung innerhalb der Sandbox setzt der Anwendung Grenzen bei der Ausführung, damit es beispielsweise nicht auf potentiell gefährdete Betriebssystemkomponenten wie das Dateisystem oder Hardware-Treiber zugreifen und auch keine systemeigenen Programme ausführen kann.

Eingeführt wurde der Mechanismus der Sandbox im Zuge der Entwicklung von Java Applets im Webbrowser. Der erste Webbrowser, der Applets ausführen konnte, war der mittlerweile eingestellte *HotJava*-Webbrowser aus dem Jahr 1997 [GWM01]. Anders als Standard-Java-Programme, müssen Applets zwingend von einer Applet-Klasse ableiten, die durch einen integrierten Sicherheitsmechanismus in einer Sandbox eingesperrt werden [Ull09]. Zunächst war die Sperre dieser Applets für alle systemkritischen Funktionen sehr strikt und undurchlässig. Danach wurde jedoch ein System von Richtlinien (sog. Policies) entworfen, das abhängig von einer gültigen Programm-Signatur eine Lockerung dieser Beschränkungen möglich macht. Sollte ein Applet unerlaubterweise diese Beschränkungen durchbrechen wollen, wird wie in Java üblich ein besonderes Ereignis ausgelöst, welches das aufgetretene Problem behandeln soll. In Java heißen diese Ereignisse auch Exceptions oder Ausnahmen. Die Standard-Richtlinie für unsignierte Java-Applets und damit die Sandbox verbietet beispielsweise explizit folgende Aktionen:

- lesende oder schreibende Operationen auf Verzeichnis- und Dateiebene,
- Netzwerkmanipulation oder Netzwerkkommunikation mit Ausnahme des Rechners, von dem das Applet heruntergeladen wurde,
- Laden und Ausführen von dynamischen Programmbibliotheken oder als nativ gekennzeichneten Methoden (das sind als sicherheitskritisch eingestufte Programmbestandteile außerhalb der JRE),
- Öffnen von sogenannten Top-Level-Fenstern (das sind Fenster, die wie eine voll funktionsfähige Standard-Java-Anwendung wirken könnten, die nicht im Webbrowser ausgeführt wird),
- Manipulation der Systemeigenschaften von Java (wie zum Beispiel den Pfad zum Heimatverzeichnis der JRE)
- Ausführung und Manipulation von anderen Programmen innerhalb des Betriebssystems und
- Beenden der JVM.

Diese Maßnahmen und andere wurden getroffen, um Java Applets von den gesamten Fähigkeiten der JRE sinnvoll abgrenzen zu können. Damit sollte eine schadhafte Wirkung durch Nutzung des Internets mittels eines Webbrowsers vermieden werden. Mittlerweile ist die Sandbox jedoch auch bei den anderen Webbrowser-Plugins wie Flash, Silverlight und dem nativen JavaScript eine wichtige Komponente, da diese Techniken aufgrund ihrer Fähigkeiten ebenso sicherheitskritisch sein können. Die massive Entwicklung von netzwerkabhängiger Client-Software führte allerdings bei Web-Anwendungen zu Aufweichungen dieser Prinzipien. Ansätze wie signiertes JavaScript [45] als Pendant zu signierten Java-Applets konnten sich bisher nicht durchsetzen. Stattdessen existieren vom Webbrowser abhängige, oft nicht standardisierte Mechanismen, um die Funktionalität in der Sandbox selbst oder darüber hinaus zu erweitern. Viele dieser Mechanismen sind derzeit Vorschläge für die Aufnahme in den bisher nicht abgeschlossenen HTML5-Standard, es ist jedoch nicht abzusehen, welche Techniken schlussendlich verfügbar sein werden.

Die Sandbox selbst ist als Software-Mechanismus durchaus nicht fehlerfrei, so dass speziell präparierte Web-Anwendungen zum Beispiel durch Ausnutzung eines Fehlers ausbrechen können. Letzte Unternehmungen, Webbrowser eng mit Grafik-Hardware und Audio-Hardware zu koppeln, um beispielsweise die Grafikdarstellung von Web-Anwendungen zu beschleunigen oder eine bidirektionale Audio-Übertragung zu ermöglichen, führten durch Fehler in der Implementation zuletzt zu solchen Ausbruchsmöglichkeiten [15].

Aufgrund der rapiden Entwicklung bei den Techniken für Web-Anwendungen ist eine genauere Betrachtung der Eigenschaften und Beschränkungen der Sandbox für installationsfreie Client-Software essentiell, um die mögliche Funktionalität der Software definieren zu können. Die von Java aufgestellten Prinzipien können nicht mehr als allgemeingültig für Web-Anwendungen bezeichnet werden, da Gegenbeispiele wie die Cross-Domain-Policy in Flash existieren. Im Einzelfall können Evaluationen für die Software-Plattform einer Web-Anwendung daher sehr kurzlebig sein.

3.4.1 Geltungsbereich und Verlassen der Sandbox

Die Sandbox gilt innerhalb des Webbrowsers für alle dort ausgeführten Plugins und natives HTML, CSS und JavaScript. Die Umsetzung der Sandbox ist webbrowserspezifisch und unterscheidet sich mitunter. Während Chrome beispielsweise für jede Website einen eigenen Prozess startet, der nicht mit den anderen kommunizieren kann, führt

Firefox die gesamte Darstellung durch einen Prozess aus, in dem auch das Sandbox-Prinzip angewendet wird. Ein Verlassen ist nur über den Mechanismus der Webbrowser-Erweiterung (siehe 2.6.4) oder durch Übergabe an eine externe Laufzeitumgebung wie Java Web Start oder AIR möglich. Die Übergabe beginnt in der Regel mit dem Download des Programms für die externe Laufzeitumgebung und anschließenden Bestätigungen zur Ausführung, die unter anderem von der Vertrauenswürdigkeit des heruntergeladenen Programms abhängen. Dies stellt ein kontrolliertes Verlassen der Sandbox dar, wohingegen durch Fehler verursachte Ausbrüche aus der Sandbox wie in [15] unkontrolliert sind und systemkritische Folgen haben können.

3.4.2 Beschränkung der Kommunikationsfähigkeit

Ein wichtiges Sicherheitskonzept der Sandbox besteht in der sogenannten Same-Origin-Policy. In Abschnitt 2.1.1 wurden Sockets als Endpunkte zweier Kommunikationspartner im Netzwerk eingeführt. Diese Definition der Endpunkte ist für eine Web-Anwendung innerhalb der Sandbox bindend, das heißt sie ist nicht in der Lage mit einem anderen Socket als dem zu kommunizieren, der die Web-Anwendung bereitgestellt hat. Diese Beschränkung kann teilweise aufgehoben werden durch Techniken wie WebSockets oder die Flash-Player-eigene Cross-Domain-Policy, bei denen zusätzliche Sockets explizit definiert werden. Auch gibt es innerhalb der HTML-Spezifikation Ausnahmen. So dürfen zum Beispiel JavaScript-Dateien und Bilder von anderen Sockets in der Web-Anwendung eingebunden werden. Dies untermauert die auftretenden Schwierigkeiten bei der Behandlung von Web-Anwendung und deren Fähigkeiten.

3.4.3 Interne Speicherschnittstellen

Speicherschnittstellen sind für jede Art von Software relevant, da es in vielen Fällen notwendig ist, Daten temporär oder persistent zu speichern. Für herkömmliche Software ist dies trivial, denn sie kann auf das Dateisystem und den Arbeitsspeicher in jedem Fall zumindest implizit zugreifen. Bei Web-Anwendungen trifft das nicht zu. Sie haben eigene Schnittstellen zur Speicherung von Daten, da der Webbrowser aus Sicherheitsgründen den Zugriff auf explizite Speicherressourcen des Betriebssystems verweigert.

Temporärer Speicher

Ein klassischer temporärer Speicher ist der Webbrowser-Zwischenspeicher (engl. auch Cache), der Grafiken und andere Inhalte von Websites speichert, um die Darstellung zu beschleunigen. Im Regelfall überprüft der Webbrowser, ob die im Zwischenspeicher enthaltenen Daten aktuell sind und empfängt andernfalls die neueren Versionen. Web-Anwendungen können den Zwischenspeicher nicht direkt nutzen, denn er dient lediglich dem Webbrowser selbst als technisches Hilfsmittel. Anders ist das bei dem für HTML5 vorgeschlagenen FileSystem-API [6], die bereits in Firefox und Chrome umgesetzt wurde. Sie ermöglicht Web-Anwendungen den temporären und persistenten Zugriff auf ein *virtuelles* Dateisystem, um dort Daten so ablegen zu können wie herkömmliche Software. Das FileSystem-API unterstützt einen Modus für temporären Speicher, dessen Größe und Inhalt die Web-Anwendung selbst festlegen kann. Der temporäre Speicher ist als solcher nicht gegen das Löschen geschützt und wird dementsprechend geleert, wenn der Nutzer den Webbrowser dazu auffordert, temporäre Dateien zu löschen.

Nichtflüchtiger Speicher

Persistente Speichermöglichkeiten sind spätestens mit dem Aufkommen von umfangreichen interaktiven Web-Anwendungen auch für den Webbrowser eine sinnvolle Erweiterung. Das FileSystem-API bietet anstelle des temporären Modus ebenso einen persistenten Modus, für dessen Gebrauch die Web-Anwendung allerdings die Zustimmung des Nutzers erfragen muss. Es handelt sich hier wiederum um ein *virtuelles* aber diesmal persistentes Dateisystem, welches nur innerhalb des Webbrowsers existiert. Andere persistente Techniken wie HTML5-AppCache [25] speichern Dateien anhand einer Index-Datei, so dass die Web-Anwendung auch noch funktionsfähig bleibt, falls die Netzwerkverbindung getrennt wird. Auch Webbrowser-basierende Datenbanken wie Indexed DB [33] existieren als Entwürfe, deren Anwendungserfolg allerdings noch nicht vorhergesagt werden kann.

3.4.4 Externe Speicherschnittstellen

Der Dateiaustausch über einen Download oder Upload wird im Webbrowser in der Regel über modale Dialogfenster exklusiv abgewickelt, das heißt der Nutzer erhält beim Download eine sperrende Nachfrage, wie mit der herunterzuladenden Datei umgegangen werden soll. Das Gleiche passiert beim Auslösen einer Aktion zum Hochladen vom

eigenen Rechner zum Web-Server. Sperrend bezieht sich in diesem Zusammenhang auf die Anwendung von modalen Dialogfenstern, die die Eigenschaft haben, den Webbrowser so lange zu blockieren bis der Nutzer eine Entscheidung getroffen hat. Dieser Zwang zum Eingreifen ist offensichtlich nicht besonders benutzerfreundlich und wird von Webbrowsern bereits aufgeweicht durch das Herunterladen im Hintergrund oder Drag-&-Drop-Funktionen.

3.5 Umsetzbarkeit des Beispiels

Die vorangegangene Evaluation der vorhandenen Techniken führt zu dem Schluss, dass vollwertige herkömmliche Software auf Computern nach wie vor zumindest einen gewissen Grad an Installationsaufwand benötigt. Dieser kann durch Prinzipien wie Portable Apps minimiert werden, was allerdings Nachteile durch zusätzlich benötigten Speicher verursacht, wenn die portable Software-Plattform zusammen mit der Client-Software ausgeliefert werden muss. In anderen Fällen müssen Voraussetzungen getroffen werden, damit jeder Nutzer die Client-Software auch benutzen kann, also wenn beispielsweise die portable Anwendung auf einem Netzwerkspeicher liegt. Webbrowser bieten sich hier als Plattform deswegen an, weil sie bereits weit verbreitet sind und der Umgang mit ihnen für viele Menschen eine alltägliche Situation darstellt. Im strikten Sinne der getroffenen Definition von installationsfreier Client-Software genügen ohnehin lediglich Web-Anwendungen den Anforderungen, auch wenn sie dafür im Gegensatz zu herkömmlicher Software Einschränkungen durch die Sandbox mit sich bringen.

3.5.1 Faktoren des Bedienkonzepts

Bei der hohen Anzahl an Internetnutzern [26] kann plausiblerweise davon ausgegangen werden, dass die Nutzung von Webbrowsern und entsprechenden Web-Anwendungen für die wenigsten Menschen eine Hürde in der Bedienbarkeit darstellt. Eine genauere Betrachtung der Web Usability wie in [Kru10] vorgenommen, kann darüber hinaus durch Tests mit Nutzern spezielle Probleme aufdecken und verhindern, die allerdings in dieser Arbeit nicht im Mittelpunkt stehen sollen. Eine klassische Web-Anwendung wird mittels einer grafischen Benutzeroberfläche mit Maus und Tastatur bedient, eventuell erscheinen dazu modale Dialogfenster beim Herunter- oder Hochladen von Dateien. Diese Vorgänge erfordern aufgrund ihrer Einfachheit keine speziellen informationstechnischen Kenntnisse und lassen sich als intuitiv für einen Internetnutzer bezeichnen.

3.5.2 Limitierungen von Web-Anwendungen

Die bereits erörterten Limitierungen von Web-Anwendungen durch das Sandbox-Prinzip verlangen einer Umsetzung technische Kompromisse ab, die bei einer nicht installationsfreien Client-Software nicht notwendig sind. Durch die Same-Origin-Policy ist die Anwendung auf die Verbindung, genauer gesagt sogar auf den gleichen Socket des Web-Servers beschränkt, der die Client-Software ausliefert. Zudem erfolgt eine Kommunikation nur über das HTTP- beziehungsweise HTTPS-Protokoll. Der direkte Zugriff auf das lokale Dateisystem des Clients ist wegen der Sandbox ebenso nicht möglich, so dass andere Speicher- und Interaktionsmöglichkeiten für Daten gefunden werden müssen. Technische Erläuterungen und Lösungskonzepte hierzu werden im Kapitel 4 untersucht und vorgestellt.

3.5.3 Auswahl einer Versionsverwaltung

Die Auswahl einer Versionsverwaltung für das zu erstellende Beispiel orientiert sich vor allem an zwei Punkten. Zum Einen ist die Interoperabilität der installationsfreien Client-Software mit der Versionsverwaltung ein Entscheidungskriterium. Zum Anderen soll im Hinblick auf die Funktion bei der SimMoLib eine sinnvolle Auswahl getroffen werden. Derzeit werden die Simulationsmodelle in der SimMoLib mittels Subversion verwaltet. Andere mögliche Versionsverwaltungssysteme wie Mercurial wurden evaluiert [Ker10]. Subversion und git wurden im Abschnitt 2.7 näher vorgestellt als Vertreter der zwei Klassen zentraler und verteilter Versionsverwaltungssysteme. Beide Systeme sind über das HTTP-Protokoll für eine Web-Anwendung ansprechbar, womit die vorgestellte installationsfreie Client-Software zunächst einmal interoperabel ist. Da jedoch Subversion bereits schon im Einsatz ist und keine technologischen Argumente für eine Änderung bestehen, werden im folgenden Kapitel daher Konzepte für eine installationsfreie Client-Software für Subversion näher untersucht.

4 Versionsverwaltung für Clients mittels Webtechnologien

Dieses Kapitel behandelt beispielhaft konkrete Techniken zur Realisierung einer Versionsverwaltung mittels installationsfreier Client-Software. Zuvor wurde bereits erörtert, dass diesen Begriff zum Zeitpunkt der Erstellung der Arbeit synonym zu Web-Anwendungen ist. Die Entwicklung eines prototypischen Demonstrators mit Hilfe von Webtechnologien bietet einen neuartigen Anwendungsbereich, der bisher mangels persistenter Speichermöglichkeiten im Webbrowser nicht umgesetzt werden konnte. Erst durch die stetige Weiterentwicklung der Webtechnologien sind Programme wie ein Client für eine Versionsverwaltung im Webbrowser realisierbar und können einen ähnlichen Funktionsumfang abdecken wie herkömmliche Software. Besonders interessant ist in diesem Zusammenhang die Frage nach dem Umgang mit der im vorigen Kapitel vorgestellten Sandbox. Auf einen Prototypen angewendete Prinzipien und Techniken sollen daher den Nachweis über die prinzipielle Möglichkeit einer Umsetzung erbringen.

4.1 Technische Konzepte zur Realisierung

Das zweite Kapitel vermittelte einen Überblick der möglichen Webtechnologien und ihrer jeweiligen Plattformen. Es wurden mehrere Techniken vorgestellt, die entweder nativ vom Webbrowser unterstützt werden wie HTML und JavaScript oder als optionale Funktionalität durch ein Plugin bereitgestellt werden wie Flash, Silverlight und Java Applets. Alle diese Techniken vereint einerseits die Beschränkung durch die Sandbox des Webbrowsers, andererseits verlangt die gegenseitige Konkurrenz eine sinnvolle Abschätzung der Umsetzbarkeit mit der jeweiligen Technik. Die Synergie von nativen Techniken des Webbrowsers und zusätzlichen Plugins erschwert eine sinnvolle Auswahl der notwendigen Techniken allerdings, denn prinzipiell sind mehrere technische Lösungswege denkbar. Flash ist zwar weit verbreitet, allerdings kein offener Standard und wird von einigen Betriebssystemen generell nicht unterstützt. Silverlight bietet aufgrund von bestimmten Systemaufrufen nur unter Windows-Betriebssystemen die volle Funktionalität und eine Steigerung der Portabilität scheint vom Hersteller Microsoft

derzeit zumindest nicht geplant zu sein [56]. Java ist universell einsetzbar und bietet durch den Signaturmechanismus die höchste Funktionalität außerhalb einer Sandbox, benötigt jedoch auch als Web-Anwendung eine vollständige Laufzeitumgebung JRE und bietet innerhalb der Sandbox keine zusätzliche Funktionalität.

Nicht eindeutig abzuschätzen sind außerdem die diesbezüglichen Weiterentwicklungen. Es ist eher anzunehmen, dass Webbrowser durch HTML5 in Zukunft ohne zusätzlich notwendige Plugins in der Lage sein werden, komplexe Anwendungen auszuführen, wie die vergangene Entwicklung von statischen Websites hin zu interaktiven Web-Anwendungen bereits gezeigt hat. Im Folgenden werden daher offene Standards beziehungsweise deren Entwürfe und native Techniken von Webbrowsern als Grundlage für das Beispiel herangezogen. Konkret bedeutet das die Beschränkung auf HTML, CSS und JavaScript. Konzeptionell sind zur Erstellung eines Clients für eine zentrale Versionsverwaltung drei Schwerpunkte zu betrachten, die folglich mit Webtechnologien umgesetzt beziehungsweise behandelt werden müssen:

1. Die Struktur der vorhandenen versionierten Daten,
2. Die Kommunikationsschnittstelle mit dem Projektarchiv,
3. Der für die Arbeitskopien genutzte persistente Speicher und der Zugriff darauf.

Algorithmen und allgemeine Software-Funktionen wie die Darstellung von Text, die Erstellung von Schaltflächen etc. können als triviale Anforderung an jede Programmierumgebung vorausgesetzt werden. Die Struktur der versionierten Daten soll zunächst einen Eindruck für den allgemeinen Anwendungszweck und die Sinnhaftigkeit einer Versionsverwaltung vermitteln. Der zweite und dritte Punkt wurden im vorigen Kapitel unter Einbeziehung der Sandbox-Eigenschaften diskutiert. Die technische Realisierung unterscheidet sich daher von klassischer Client-Software und soll in den folgenden Abschnitten näher erläutert werden, um auch die Prinzipien und Unterschiede von Web-Anwendungen gegenüber klassischer Client-Software deutlich zu machen.

4.1.1 Struktur der versionierten Daten

Auch wenn das zurückliegende Kapitel die Nutzbarkeit eines auf Webtechnologien basierenden Clients im Rahmen des Projekts SimMoLib eingeschränkt hat, ist die generelle Betrachtung der möglichen Technologien nach wie vor lohnenswert. Daher soll im Folgenden davon ausgegangen werden, es gäbe für SimMoLib einen Anwendungsbereich, in dem eine kollaborative Arbeit an versionierten Dokumenten wie zum Beispiel Textdateien für Metadaten innerhalb des Webbrowsers sinnvoll ist. Die Funktions-

und Arbeitsweise einer Versionsverwaltung wird durch die Beschränkung auf eine Bearbeitung im Webbrowser ohnehin nicht berührt.

Ein in SimMoLib vorhandenes Simulationsmodell ist in einer definierten Ordner- und Dateistruktur organisiert und auf diese Weise auch im Versionskontrollsystem abgelegt. Das Root-Verzeichnis von Subversion bildet gleichzeitig das Basis-Verzeichnis für SimMoLib. Es umfasst das komplette Projektarchiv und damit die gesamte Bibliothek für Simulationsmodelle. Das Projektarchiv ist über einen URI im Netzwerk und über HTTP als Verbindungsprotokoll erreichbar. Im Projektarchiv werden Nutzerordner mit eindeutigen Identifikationsnummern angelegt, so dass der Teilpfad ab diesem Ordner nur für den jeweiligen Nutzer von SimMoLib gültig ist. Innerhalb des Nutzerordners wiederum befinden sich einzelne Ordner für je ein Simulationsmodell und ein Ordner `__releases__`, der komprimierte Datei-Archive enthält, die jeweils einer stabilen und veröffentlichten Version eines Simulationsmodells entsprechen.

Der Modellordner selbst enthält eine definierte Struktur aus Ordnern und Dateien wie zum Beispiel einen Ordner `doc` für die Dokumentation oder einen Ordner `src` für Quelltexte, sowie Metadaten für das Simulationsmodell in einer `.project`-Datei. Eine weitere Verzweigung ist wie bei üblichen Dateisystemen möglich und vorhanden. Der Nutzer ist bei dem durch Subversion realisierten Projektarchiv in der Lage, unterschiedliche Teilbäume dieser Datenstruktur rekursiv auszuchecken. Das bedeutet, er kann den kompletten Inhalt des Projektarchivs als Arbeitskopie herunterladen oder auch nur einen einzelnen Modellordner. Darüber hinaus ist der Inhalt eines Modellordners beliebig und enthält Dateien mit proprietären Datenformaten von Office- oder Simulationsprogrammen, sowie text-basierte Dokumente. Für eine Versionsverwaltung ist lediglich die Unterscheidung zwischen Binär- und Text-Format relevant, da beide Formate in der Regel unterschiedliche Algorithmen benötigen, wenn vorgenommene Änderungen erkannt werden sollen.

Um die Aspekte der Funktionalität des Beispiels zu beleuchten, genügt es, sich in den folgenden Abschnitten technisch auf einen einzelnen Modellordner zu konzentrieren. Zum Einen ist ein Simulationsmodell die zentrale logische Entität innerhalb von SimMoLib und zum Anderen entstehen dadurch keine Einschränkungen, denn die Nutzung von Subversion ist in jeder Ordnerebene durch die Rekursion absolut gleichwertig. Zudem werden Revisionen in Subversion global für das gesamte Projektarchiv zugewiesen. Ändert sich eine Datei in einem beliebig verzweigten Pfad, bekommt dieser explizite *Zustand des gesamten Projektarchivs* eine neue Versionsnummer [14]. Auch deswegen ist es ausreichend, die Arbeitsweise mit einem beliebigen Unterordner des Projektarchivs zu erörtern. Generell müssen bei einer Versionsverwaltung zwei unterschiedliche

Domänen bei der Behandlung der vorhandenen Daten unterschieden werden. Ein Simulationsmodellordner ist im Projektarchiv inklusive aller bisherigen Versionen vorhanden. Zusätzlich befindet er sich als Arbeitskopie lokal in einem Dateisystem oder einem anderen Speichersystem. Die Korrelation zwischen beiden Domänen ist deswegen möglich, weil die Ordner- und Dateistruktur beim Checkout erhalten bleibt. Die Arbeitskopie hält außerdem die heruntergeladene beziehungsweise aktuelle Revision in einer Metadaten-Struktur vor, um die in der Folge stattfindenden lokalen Änderungen damit vergleichen und erkennen zu können.

4.1.2 Persistente Speicherung durch das FileSystem-API

Die Arbeitsweise von Versionsverwaltungssystemen verlangt in der Regel nach nichtflüchtigem Sekundärspeicher wie im Abschnitt 3.1.2 beschrieben, da Änderungen in lokalen Arbeitskopien beim Wegfall der Energieversorgung des Client-Rechners verloren gingen. Jede Software, die lokal Daten vorhält und ändert, sollte diese vor allem zur Sicherung in nichtflüchtigem Speicher ablegen können. Die Untersuchung von installationsfreier Client-Software und ihren eingeschränkten Fähigkeiten durch die Sandbox ergab jedoch, dass der direkte Zugriff auf das Dateisystem des Betriebssystems nicht möglich ist. Dennoch wird nichtflüchtiger Speicher mittlerweile auch bei umfangreichen Web-Anwendungen zum Beispiel zur Beschleunigung der Darstellung oder zum Zwischenspeichern von Nutzerdaten eingesetzt. Beispiele für solche Speicherschnittstellen wurden im Abschnitt 3.4.3 vorgestellt. Das von dem FileSystem-API erzeugte *virtuelle* Dateisystem für Web-Anwendungen unterscheidet sich im Aufbau und in der Struktur nicht von dem *realen* Dateisystem des Betriebssystems [6]. Der hier realisierte Subversion-Client nutzt daher diese Schnittstelle zur Speicherung der Arbeitskopien.

Das FileSystem-API implementiert technisch gesehen ein Dateisystem-Objekt einer bestimmten Größe in JavaScript, das anschließend manipuliert werden kann. Dem Dateisystem-Objekt können wiederum untergeordnete Objekte wie Dateien und Ordner hinzugefügt werden, die wie bei bekannten Dateioperationen verändert oder gelöscht werden können. Prinzipiell unterscheidet das FileSystem-API zwei Modi zur Definition von persistenten Daten. Ein mit der Eigenschaft *TEMPORARY* angelegtes virtuelles Dateisystem kann vom Webbrowser ohne Einwilligung der Web-Anwendung oder des Nutzers gelöscht werden, zum Beispiel beim Leeren des Webbrowser-Caches. Es besteht also keine Garantie, dass die Daten beim nächsten Start der Web-Anwendung noch vorhanden sind. Im Gegensatz dazu soll ein Dateisystem mit der Eigenschaft *PERSISTENT* nicht vom Webbrowser gelöscht werden, wohl aber vom Nutzer oder der

Web-Anwendung selbst. Beim Anlegen eines persistenten virtuellen Dateisystems soll der Webbrowser beim Nutzer nach der Berechtigung für das von der Web-Anwendung angeforderte Speicherlimit (auch *Quota* genannt) fragen [6]. Bevor der Webbrowser diesen permanenten Speicher anlegt, erscheint daher eine Aufforderung zur Bestätigung der Speichergröße, die dann als Parameter einer Funktion an den Befehl zur Erstellung des Dateisystem-Objekts weitergegeben wird [6]. Anschließend kann die Web-Anwendung das virtuelle Dateisystem nutzen.

4.1.3 REST-Prinzip und Same-Origin-Policy

Den komplementären Aspekt der Kommunikation auf Client-Seite, also die Fähigkeit mit Subversion sinnvoll Daten auszutauschen, erfüllt eine in JavaScript programmierte Komponente, die mit Hilfe von HTTP-Anfragen einerseits Daten empfängt und andererseits Daten sendet [29]. Grundlage dieser Kommunikation bildet das von Roy Fielding geprägte Konzept des Representational State Transfer (REST) [Fie00]. Der REST beschreibt eine im Internet typische Art und Weise der Kommunikation und ihres Designs von Clients und Servern im Netzwerk. Im Einzelnen bedeutet das:

- durch URIs eindeutig definierte *Ressourcen* bilden die Quellen für Informationen,
- Client und Server tauschen gegenseitig *Zustände* dieser Ressourcen über ein *Interface* wie HTTP und dessen unterstützte Methoden wie GET, PUT, POST, usw. aus,
- die Ressourcen liegen in gängigen Austauschformaten (des Internets) wie JSON oder XML vor,
- Client und Server sind möglichst unabhängig voneinander skalierbar und benötigen keine expliziten Informationen über die Beschaffenheit des jeweils anderen Kommunikationspartners.

Subversion kann als sogenannter *RESTful Web Service* bezeichnet werden, da es obige Kriterien erfüllt, wobei der Client mit Hilfe der für Subversion definierten HTTP-Methoden Anfragen an das Repository beziehungsweise dessen URIs stellt. Die Anfragen werden vom Subversion-Server mit einem Status-Code und dem dazugehörigen Inhalt beantwortet. Der Inhalt kann zum Beispiel aus Struktur-, Meta- oder versionierten Daten des Repositorys bestehen. Abbildung 4.1 visualisiert beispielhaft eine REST-basierte Kommunikation zwischen dem Subversion-Server und dem Subversion-Client.

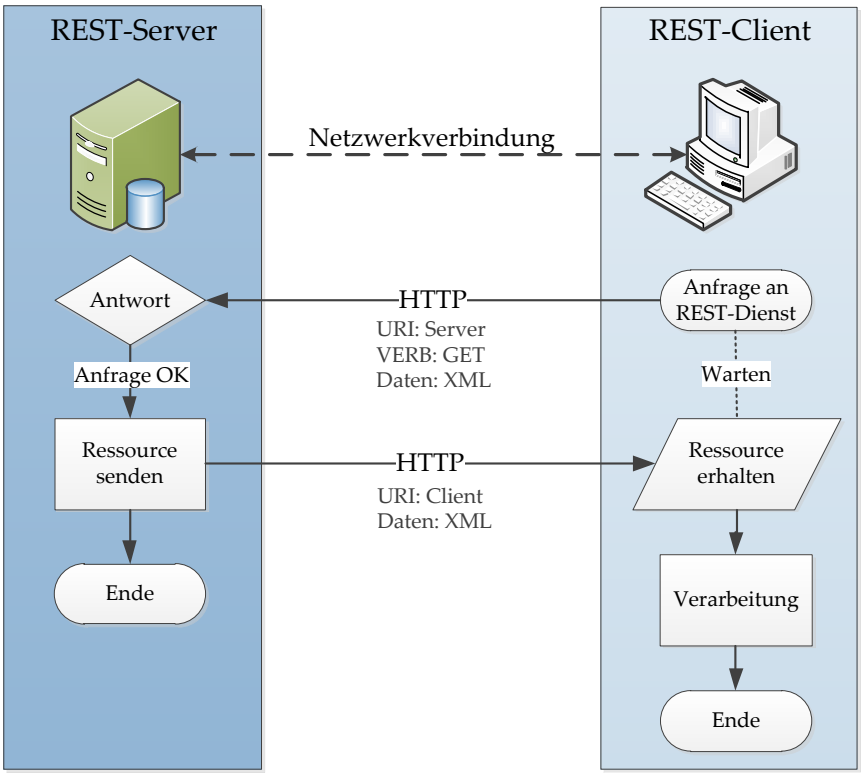

Abbildung 4.1: Beispiel einer REST-basierten Kommunikation

Für Web-Anwendungen wie den hier erstellten Versionsverwaltungs-Client gilt die Same-Origin-Policy fast ausnahmslos. Das erfordert in der Regel eine spezielle Infrastruktur auf dem Web-Server. Er muss sämtliche Dienste über *einen einzigen Socket* anbieten können und gleichzeitig auch die Web-Anwendung ausliefern. Bisher wird für asynchrone Anfragen an den Web-Server (das sogenannte AJAX) ein Daten-Objekt namens *XMLHttpRequest* genutzt, das der Same-Origin-Policy unterliegt und Daten nur mit dem Socket des Web-Servers austauscht, der die Web-Anwendung auch ausgeliefert hat [29]. In einem Entwurf für HTML5 unterstützt dieses Objekt auch die Kommunikation mit anderen Sockets durch das Konzept des *Cross-Origin Resource Sharing (CORS)* [30, 31], was allerdings noch nicht flächendeckend unterstützt wird.

Derzeit werden Web-Server daher oft so angepasst, dass der Client die Same-Origin-Policy einhalten kann. Das ist beispielsweise durch Proxy-Server möglich, die zwischen Client und Server platziert werden und transparent für den Client die Daten von verschiedenen Servern weiterleiten oder durch URI-basierte Techniken, bei denen `www.beispiel.de/web/` die Web-Anwendung und `www.beispiel.de/db/`

die Datenbank ausliefert, so dass auf dem Web-Server zwei verschiedene Prozesse Daten verarbeiten und empfangen beziehungsweise senden, aber beide diese Dienste nur über einen gemeinsamen Socket zur Verfügung stellen. Das bereits in Abschnitt 2.7.2 vorgestellte Subversion bietet diese Möglichkeit ebenfalls mit Hilfe des erwähnten Web-Server-Moduls, so dass der Web-Server gleichzeitig die Web-Anwendung ausliefert und Zugriff auf die Versionsverwaltung von Subversion ermöglicht.

4.2 Entwicklung eines prototypischen Demonstrators

Im Rahmen dieser Diplomarbeit wurde am DLR ein prototypischer Demonstrator umgesetzt, der als installationsfreie Client-Software den Zugriff auf SimMoLib bei einer konkreten Umsetzung erlauben würde. Dazu wurde ein Subversion-Test-Repository angelegt und die zuvor beschriebene Datenstruktur nachgebildet. Eine kombinierte grafische Benutzeroberfläche als Browser für das Projektarchiv und für die Arbeitskopien, sowie als Editor für Text-Dateien bietet visuellen und interaktiven Zugriff auf SimMoLib. Durch die nähere Betrachtung von Hintergründen und Arbeitsabläufen, sollen dem Leser die vorgestellten Techniken einer Web-Anwendung anschaulich gemacht werden.

4.2.1 Entwurf einer Benutzeroberfläche

Zuvor wurde bereits die Unterscheidung bei einer zentralen Versionsverwaltung zwischen der Ebene des Projektarchivs und der Ebene der Arbeitskopie deutlich gemacht. Beide Ebenen ermöglichen andere Arbeitsabläufe und -vorgänge, was ebenfalls im grafischen Entwurf zur Geltung kommen soll. In der Arbeitskopie können Dateien und Ordner gelöscht, erstellt und verändert werden. Die Oberfläche für die Arbeitskopie verhält sich also prinzipiell wie ein Datei-Browser mit Editierfunktionen. Die Oberfläche für das Projektarchiv hingegen bietet das Durchsuchen, Herunterladen und Anzeigen der enthaltenen Daten. Der Fokus wird hierbei zweckmäßig auf die Simulationsmodelle gelegt, die als Ordner zur Verfügung stehen. Diese werden in einer Ebene unterhalb eines Nutzer angelegt und auch dort verwaltet. Um die Arbeit mit dem Client für den Nutzer übersichtlicher zu gestalten, werden daher nur komplette Simulationsmodelle als Arbeitskopie ausgecheckt. Abbildung 4.2 zeigt die Nutzung des Projektarchivs mittels einer Datei-Browser-ähnlichen grafischen Oberfläche, die technisch durch die in Abschnitt 4.1.3 näher beschriebene Funktionalität für Subversion in JavaScript umgesetzt wurde. Die Kommunikation des Subversion-Clients mit dem Subversion-Server

wird realisiert durch asynchrone und ereignisgesteuerte Datenübertragung, so dass nur die aktualisierten Teile der Benutzeroberfläche nachgeladen werden müssen, wenn der Nutzer beispielsweise einen Modellordner durchsucht.

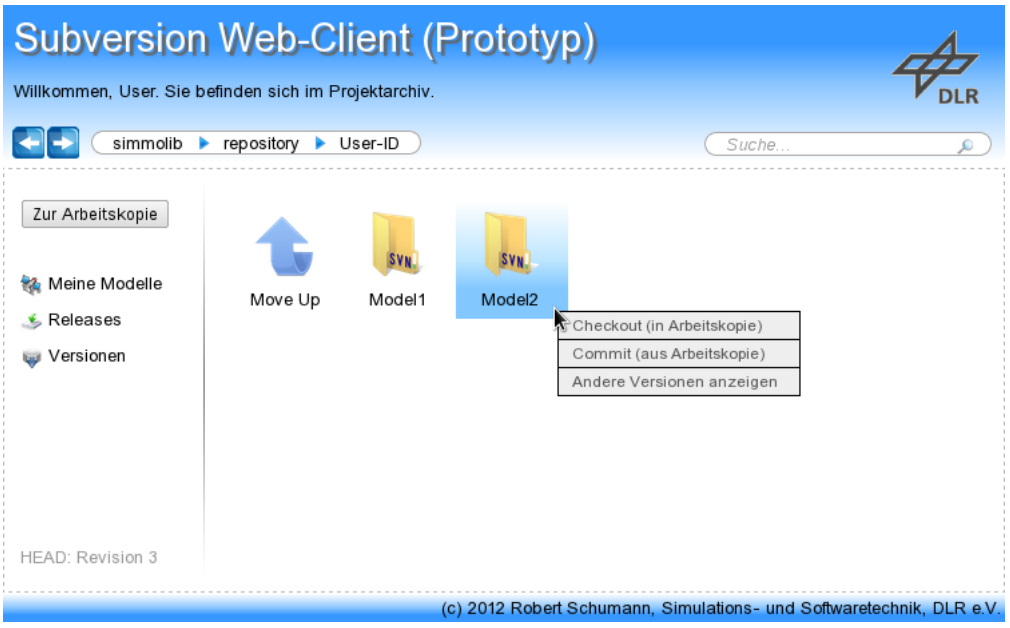

Abbildung 4.2: Prototyp der Projektarchiv-Ansicht

Eine farblich auffällige Unterscheidung zwischen der Projektarchiv-Ansicht (siehe Abbildung 4.2) und der Arbeitskopie-Ansicht (Abbildung 4.3) soll dem Nutzer der Client-Software beiden unterschiedlichen Arbeitsgebiete visualisieren und verdeutlichen, dass beide Teile der Oberfläche jeweils eigene Optionen für die dort dargestellten Inhalte bieten. So ist es möglich, die Simulationsmodelle in der Arbeitskopie-Ansicht direkt inhaltlich zu bearbeiten oder sogar deren Erstellung, Löschung etc. vorzunehmen. Die inhaltliche Bearbeitung ist derzeit auf Text-Dateien begrenzt, da die Unterstützung von proprietären Binärformaten innerhalb einer einzigen Web-Anwendung nicht im Fokus dieser Arbeit steht. Für den Fall, dass solche Dateien vorliegen und bearbeitet werden müssen, ist ein manueller Download beziehungsweise Upload in der Arbeitskopie-Ansicht nach der Bearbeitung im lokalen Dateisystem bisher unumgänglich. Das Problem proprietärer Binärdaten ist allerdings in allen Betriebssystemen ein prinzipielles. Die Unterstützung hierfür muss in der Regel durch externe Programme bereitgestellt werden, die nicht in das Betriebssystem integriert sind. Es existieren allerdings auch für Desktop-Anwendungen mehrfach Ansätze, im Sinne von REST austauschbare Datenformate wie XML zu verwenden. Während Microsoft-Office-Dateien beispielsweise lange ein Binärformat hatten, werden sie heute für gewöhnlich in einem portabel komprimierten XML-Austauschformat abgespeichert, so dass eine Interpretation der Informationen

leicht zu bewerkstelligen ist. Für beide Oberflächen sollen im Folgenden konkrete Anwendungsfälle mit Bezug zur technischen Umsetzung kurz beschrieben werden.

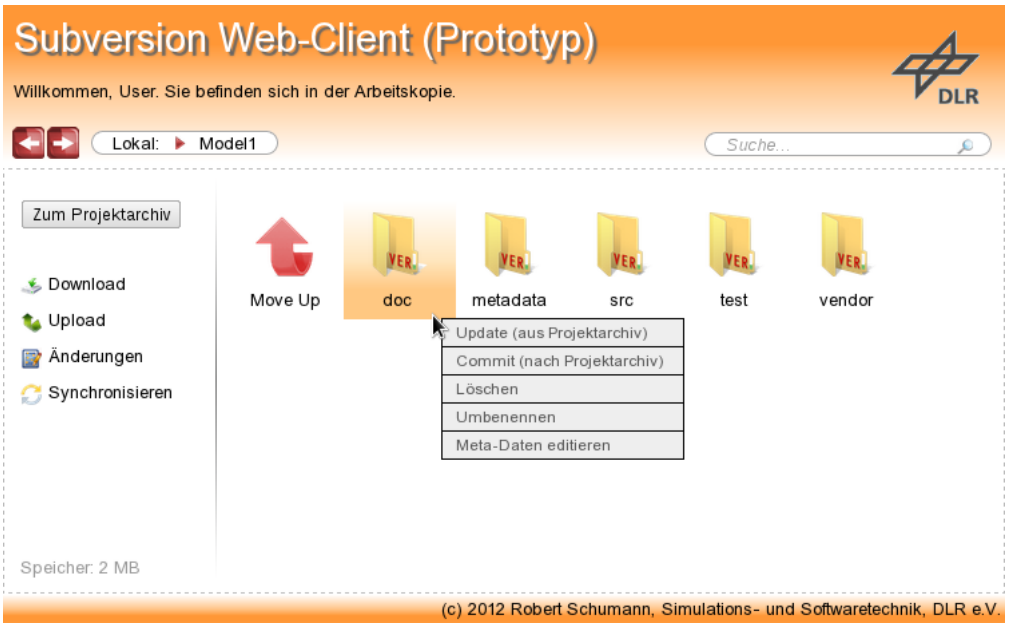

Abbildung 4.3: Prototyp der Arbeitskopie-Ansicht

4.2.2 Das Projektarchiv betreffende Anwendungsfälle

Die Interaktion und Kommunikation mit dem Projektarchiv von Subversion erfolgt über JavaScript-Funktionen, die auf dem darin vorhandenen `XMLHttpRequest`-Objekt [29] aufbauen. Dieses Objekt bietet vereinfacht gesagt, die in Abschnitt 4.1.3 beschriebene Funktionalität einer REST-Kommunikation mit dem Subversion-Server und unterliegt der Same-Origin-Policy. Es besitzt Eigenschaften und Methoden, die gesendete und empfangene Daten enthalten und verarbeiten und von der Änderung des Verbindungszustands ausgelöst werden. Realisiert wird das über eine Funktionsreferenz für einen sogenannten Event-Handler des Objekts. Ändert sich bei dem Vorgang der Kommunikation mit dem Subversion-Server etwas, also werden beispielsweise Daten oder eine Fehlermeldung empfangen, löst der Event-Handler `onReadyStateChange` des Objekts die Funktion aus, welche nachfolgend auf die Änderung reagiert. Die weitere Verarbeitung beinhaltet in der Regel eine Abfrage des Rückgabewertes des Subversion-Servers. HTTP-Rückgabewerte wie 404 für einen nicht gefundenen URI oder 403 für einen verbotenen Zugriff sind auch dem Nutzer eines Webbrowsers meist bekannt. Der Wert 200 für eine erfolgreiche Anfrage erlaubt es, die im `XMLHttpRequest`-Objekt empfangenen Daten auszuwerten und zu verarbeiten, welche XML-strukturiert vorliegen [19]. Die beschriebene Funktionalität wird im Demonstrator durch eine eigens

erstellte JavaScript-Bibliothek namens `svn_client.js` bereitgestellt und realisiert damit die notwendige Kommunikation der Client-Software mit der Server-Software Subversion.

Subversion besitzt viele Funktionen im Rahmen seines Anwendungsprotokolls, das aus HTTP, WebDAV und DeltaV besteht. Da deren Implementation allerdings prinzipiell sehr ähnlich verlaufen würde, ist die folgende Auswahl an vorgestellten Anwendungsfällen bereits repräsentativ und in keiner Weise als Einschränkung zu verstehen.

Checkout

Unter dem Prozess des Checkouts wird das Herunterladen einer Ordner- und Dateistruktur einer Versionsverwaltung als lokale Arbeitskopie verstanden. Im Beispiel wird dabei der Einfachheit halber von einem Simulationsmodellordner ausgegangen. Die lokale Arbeitskopie wird im virtuellen Dateisystem des Webbrowsers angelegt, das wie gehabt als JavaScript-Objekt zuvor erstellt werden muss. Die Erstellung des Dateisystem-Objekts `requestFileSystem` hängt dabei maßgeblich von dem unter 4.1.2 beschriebenen Vorgang ab. Nach der Initialisierung enthält das Objekt ein untergeordnetes Objekt namens `root`, das das Wurzelverzeichnis des virtuellen Dateisystem repräsentiert. Dieses Objekt besitzt wiederum Methoden zur Erstellung, Bearbeitung und Löschung von Dateien und Ordnern. Die Funktion `getFile()` legt beispielsweise eine Datei an, wenn sie noch nicht existiert und erstellt damit ihrerseits ein neues Objekt vom Typ `FileEntry`. FileEntry- und DirectoryEntry-Objekte unterhalb von `root` stellen die eigentlich Dateien und Ordner des virtuellen Dateisystems dar. Darüber hinaus existieren Funktionen zur Abfrage und Manipulation von Dateiinhalten über sogenannte FileReader und FileWriter-Schnittstellen. Die Funktionalität des virtuellen Dateisystems setzt der hier vorgestellte Demonstrator in einer JavaScript-Bibliothek `svn_fs.js` um.

Bei der lokalen Speicherung einer Arbeitskopie werden außerdem zusätzliche Metadaten angelegt, die eine lokale Bearbeitung nachvollziehbar und indizierbar machen. Während eines Checkouts erstellt der Client wie auch bei klassischen Subversion-Clients einen versteckten Ordner namens `.svn` in jedem im virtuellen Dateisystem erstellten Ordner der Arbeitskopie. Die Metadaten in diesen Ordnern enthalten die heruntergeladene Revision ohne Änderungen, damit der Client bei Änderungen der Dateien der Arbeitskopie die Unterschiede berechnen und dann ebenfalls innerhalb der Metadaten abspeichern kann. Die Struktur der Metadaten ist in [14] näher beschrieben. Der Client kapselt die Funktionalität der Metadaten mit Hilfe eines Objekts in der dafür

angelegten Bibliothek `svn_wc.js`, das sich auf einen Pfad im virtuellen Dateisystem bezieht. Da die Metadaten ebenfalls als strukturierte Ordner und Dateien vorliegen, greift `svn_wc.js` bei der Erstellung auch auf die Dateisystem-Funktionen von `svn_fs.js` zurück.

Durch die drei vorgestellten JavaScript-Bibliotheken kann damit ein Checkout realisiert werden. Technisch gesehen fragt der Client über das `XMLHttpRequest`-Objekt beispielsweise zunächst die aktuelle Revision des Projektarchivs und den Pfad dazu über den URI des Subversion-Repositorys und dem HTTP-Verb `OPTIONS` ab, woraufhin er mit jenem Pfad und der Anfrage des HTTP-Verbs `REPORT` rekursiv den Inhalt des Pfades im Projektarchiv zurückgeliefert bekommt. Im virtuellen Dateisystem erstellt er aus den erhaltenen Daten rekursiv die Ordner- und Dateistruktur inklusive der Metadaten. Angestoßen wird der Checkout über das Kontextmenü in der grafischen Benutzeroberfläche, welche sich in der Arbeitskopie entsprechend ändert.

Update

Nachdem ein Nutzer eine Arbeitskopie ausgecheckt hat, kann es vorkommen, dass andere Nutzer Änderungen in das Projektarchiv übertragen, die die vom Nutzer ausgecheckten Dateien und Ordner betreffen. Für diesen Fall existiert der Mechanismus des Updates einer Arbeitskopie. Der Client vergleicht hierbei die im virtuellen Dateisystem vorhandenen Daten mit der Revision des Repositorys und versucht die Unterschiede zusammenzuführen. Es ist leicht zu erkennen, dass wiederum alle vorgestellten Bibliotheken des Clients benötigt werden: zur Kommunikation mit dem Subversion-Server, zur Verwaltung des virtuellen Dateisystems und zur Bearbeitung der Arbeitskopie-Metadaten. In diesem Zusammenhang ist allerdings zu erwähnen, dass ein Update durchaus fehlschlagen kann, wenn Änderungen nicht zusammengeführt werden können. Diese Merge-Konflikte sind höchst problematisch und verhindern geradezu eine simple Nutzung der Versionsverwaltung. Ausführlich behandelt wurden solche Konflikte und mögliche Algorithmen zur Auflösung in [Men02].

Commit

Beim Prozess des Commits vergleicht der Client die in der Arbeitskopie vorhandenen Dateien und Ordner mit denen der Metadaten. Er generiert daraus Differenzen, falls Unterschiede vorhanden sind und sendet diese wiederum mittels der Funktionen in `svn_client.js` zum Subversion-Server. Das Format der Differenzen basiert auf einem

simplen Algorithmus von Subversion, der in der Bibliothek `svndiff.js` definiert ist und den der Client nutzt. Die Implementation mittels Webtechnologien wird im Abschnitt 4.2.4 genauer erläutert. In der Regel muss der Nutzer für einen Commit auch einen Kommentar in die Versionsverwaltung einfließen lassen, damit eine menschlich verständliche Formulierung der Änderungen verfügbar ist. Möchte ein Nutzer exklusiv an einer Datei arbeiten und diese dann per Commit aktualisieren, gestattet es Subversion, die Datei global im Projektarchiv durch ein sogenanntes Lock zu sperren. Andere Nutzer können diese Datei dann so lange nicht verändern bis der Nutzer die Datei wieder entsperrt. Technisch gesehen funktioniert dieser Schritt über einen nutzerbasierten Token, der im Projektarchiv erstellt wird und auf die Sperre eines Nutzers verweist. Die Vorgehensweise beinhaltet abermals Funktionen von HTTP-Verben wie `LOCK` und `UNLOCK` (vgl. dazu auch [19]), es wurde aber im Demonstrator aufgrund der höheren Komplexität nicht umgesetzt.

4.2.3 Die Arbeitskopie betreffende Anwendungsfälle

Für die Bearbeitung der eigentlichen Dateien und Ordner eines ausgecheckten Simulationsmodells ist die Benutzeroberfläche der Arbeitskopie des Demonstrators gedacht. Sie spiegelt den Inhalt des virtuellen Webbrowser-Dateisystems wider und bietet dem Nutzer Interaktionsmöglichkeiten auf dieser Ebene. Hervorzuheben ist bei dieser Art der Realisierung der Unterschied zu einem herkömmlichen Client. Das virtuelle Dateisystem ist nur dem Webbrowser zugänglich, was zunächst wie ein Nachteil wirkt, da ein einfacher Datenaustausch mit anderen lokal ausführbaren Programmen nicht möglich ist. Allerdings bietet die Kombination der Funktionalität eines Subversion-Clients *und* einer Bearbeitungsoberfläche für die Dateien und Ordner eine einfache und bei korrekter Implementation mitunter für den Nutzer eine weniger fehlerträchtige Software-Gesamtlösung.

Begründen lässt sich dies mit der Tatsache, dass herkömmliche Subversion-Clients die Arbeitskopie im Dateisystem des Rechners erstellen. Dort sind durch viele Einflüsse Änderungen möglich, denn der Nutzer und auch die Programme des Rechners respektieren unter Umständen nicht, dass die vorliegenden Dateien und Ordner fest durch Subversion strukturiert sind. Änderungen in der Struktur, also beispielsweise Erstellen eines Ordners, Löschen einer Datei, Verschieben einer Datei usw. sind jedoch Aufgaben, die vom Subversion-Client ausgeführt werden müssen, damit dieser die Metadaten korrigiert und die Änderungen auch in das Projektarchiv übernehmen kann. Das bedeutet im Dateisystem des Rechners muss beispielsweise das Kommando `svn`

`rm datei` ausgeführt werden, um eine Datei zu löschen und nicht `rm datei`[1], da nur Ersteres den Subversion-Client über die Kommandozeile anspricht und korrekt instruiert. Im Dateisystem des Betriebssystems existiert dafür keine Zwangsvorrichtung, so dass jedes Programm mit Zugriff auf das Dateisystem anstelle des Subversion-Clients diese Operationen vornehmen kann. Subversion besitzt Heuristiken, um bestimmte Vorgänge zu erkennen, aber es würde im Projektarchiv beispielsweise niemals eine Datei löschen, die im Dateisystem ohne seine Kenntnis gelöscht wurde, was zu inkonsistenten Datenbeständen führen kann.

Bearbeitung eines Dokuments

Da das FileSystem-API eine neue Technologie im Rahmen der Entwicklung von HTML5 ist, muss die Bearbeitung von Dokumenten im Webbrowser erst daraufhin ausgerichtet werden. Standard-Editoren für Dateien im Webbrowser senden in der Regel nach der Bearbeitung des Dokuments ein sogenanntes Formular an den Web-Server ab, der es serverseitig mittels PHP oder anderen Programmiersprachen verarbeitet. Ein Formular ist vereinfacht gesagt nichts anderes als ein Datensatz aus Argumenten und Werten, wobei das auch Binärdaten nicht ausschließt. Der Demonstrator hingegen lädt und speichert über ein rudimentäres HTML-Text-Eingabefeld die im virtuellen Dateisystem vorhandene Datei, wenn der Nutzer sie bearbeitet. Da es sich hier um einen Prototypen handelt, ist die Programmierung eines vollständigen Text-Editors hier nicht von Relevanz, aber mit entsprechendem Aufwand problemlos möglich, was JavaScript-Editoren wie TinyMCE beweisen [51]. Die Funktionalität des Editors ist in der JavaScript-Bibliothek `text_editor.js` des Demonstrators enthalten.

Dateioperationen auf Dokumenten

Wie bereits erläutert müssen Änderungen in Datei- oder Ordner-Namen oder Verschiebungen vom Subversion-Client durchgeführt werden. Im Fall des FileSystem-API ist das im virtuellen Dateisystem ebenso einfach wie bei einem herkömmlichen Client. Das schon beschriebene FileEntry-Objekt für eine Datei besitzt Methoden wie `moveTo()` und `copyTo()`, die Parameter für das Ziel-Verzeichnis und optional für einen neuen Dateinamen auswerten. Methoden innerhalb von `svn_wc.js` ändern zeitgleich die

[1] `rm` ist ein Standard-Kommando für Dateisysteme unter Unix und Linux und die Kurzform für das englische Wort *remove*

Metadaten der Arbeitskopie, so dass die Subversion-Struktur fehlerfrei bleibt. Prinzipiell sind so alle Dateioperationen denkbar, die auch normale Dateisysteme bieten, da einerseits das FileSystem-API die Operationen unterstützt und die Bibliotheken des Demonstrators dazu benutzt werden können, die Metadaten der Arbeitskopie zu synchronisieren. Ein Fremdzugriff anderer Software ist nicht möglich, womit der vorgestellte Demonstrator auch Vorteile gegenüber klassischen Versionsverwaltungs-Clients besitzt.

4.2.4 Subversion-Algorithmus für einen differenziellen Dateivergleich

Algorithmen für differenzielle Dateivergleiche sind für die Programmierung allgemein und im Besonderen für Versionsverwaltungen eine Voraussetzung, um einerseits Unterschiede zwischen Dateien sichtbar zu machen und diese andererseits auch effizient transportieren zu können. In der Entwicklung von Computerprogrammen werden häufig nur diese Unterschiede, auch *Patches* genannt, übertragen, evaluiert und auf die Quelltexte angewendet, da eine Übertragung der gesamten Datei oft ineffizient ist, wenn nur wenige Änderungen an ihr vorgenommen wurden. Prinzipiell wird bei der Erstellung solcher Differenzen zunächst einmal zwischen Text- und Binärdateien unterschieden. Text-Dateien können im Gegensatz zu Binärdateien zeilenbasiert miteinander verglichen werden. Jedes Wort der Zeile lässt sich durch Editieren der einzelnen Buchstaben in ein anderes Wort transformieren. Die Buchstaben selbst sind in der Regel durch einen ganzzahligen Wert unterschiedlicher Größe innerhalb eines Zeichensatzes des Betriebssystems definiert, wobei ein großes A beispielsweise den Wert 65 annimmt und ein kleines a den Wert 97 [11]. Die Differenz der beiden Werte bildet ein Ähnlichkeitsmaß für die beiden Buchstaben a und A, so dass damit eine sogenannte Editierdistanz für die Transformation eines Wortes in ein anderes, auch Levenshtein-Distanz genannt, berechnet werden kann [Lev66]. Die Editierdistanz der Wörter *BAr* und *Bar* beträgt demnach beispielsweise den Wert 32. Die Berechnung dieser Distanz und die Nutzung von Matrizen zum Vergleich zweier Wörter lässt zusätzlich andere Schlüsse zu. So ist es zum Beispiel möglich zu erkennen, dass eine Zeile nur innerhalb der Textdatei verschoben wurde. Eine Umsetzung von solchen Algorithmen und Effizienzbetrachtungen wurden vielfältig vorgenommen wie beispielsweise in [Mye86]. Ein verbreitetes Standard-Programm zur Erstellung von austauschbaren Patches im Standard-Format Ünified Diffist `diff` nach einem Algorithmus aus [HM76]. Viele Formate für Dateidifferenzen und die dazugehörigen Programme nutzen daher ebenso diff als Präfix oder Suffix. Die explizite Erkennung von Zeichen-Änderungen in Binärdateien ist hingegen ungleich schwieriger, da weder einzelne Zeilen, noch einzelne

Wörter separiert werden können. Dennoch existieren auch hierfür Algorithmen.

Subversion nutzt zur Berechnung von Differenzen einen eigenen Algorithmus, der auch auf Binärdateien angewendet und durch JavaScript-Funktionen im Demonstrator genutzt werden kann. Der Unterschied zwischen zwei Dateien wird bei Subversion als sogenanntes Delta bezeichnet und in einem speziellen Dateiformat als SVN-Delta abgespeichert [2]. Ein SVN-Delta besteht wie in Abbildung 4.4 aus einer identifizierenden Zeichenkette am Anfang der Datei, auch *Header* genannt und einer wohl definierten Sequenz von beliebig vielen *Windows*. Jedes Window enthält Informationen über Anfangspunkt, Länge und Änderungswerte von bestimmten Bereichen in einer Zieldatei, wodurch es die Quelldatei zur Zieldatei transformieren kann. Die einzelnen Änderungen werden dabei in *Instructions* gekapselt, die in etwa eine Überführungsfunktion darstellen. Windows überlappen sich in ihren definierten Bereichen nicht. Ist es zusätzlich notwendig, neue Daten in die Quelldatei bei der Transformation einzufügen, werden diese Daten innerhalb der *New Data*-Sektion des Windows gespeichert und mit der Differenz übertragen. Die New-Data- und Instructions-Sektionen können als komprimierte Datenblöcke vorliegen [2].

Die Übertragung der SVN-Deltas vom Subversion-Server zum Demonstrator und vice versa erfolgt als Base64-codierte Dateneinheiten. Base64 ist ein verbreitetes Austauschformat für Daten im Netzwerk, was beispielsweise auch bei der Versendung von Mail-Anhängen genutzt wird. Es reduziert die zur Informationserhaltung der übertragenen Daten notwendigen Zeichen auf einen lesbaren Zeichensatz, der nur Buchstaben, Zahlen und wenige Sonderzeichen enthält [28]. Dadurch verursachen auch Binärdaten, die nicht darstellbare Zeichen enthalten, oder spezielle Steuerzeichen keine Probleme bei der Verarbeitung. Allerdings vergrößert sich aufgrund der ineffizienteren Codierung auch der Speicherbedarf für dieselbe Menge an Information. Der Empfänger muss die erhaltenen Daten wieder in das eigentliche Format dekodieren.

Ein Window bezieht sich jeweils auf zusammenhängende Sektionen in den verglichenen Dateien. In der Quelldatei wird dieser Bereich *Source View* genannt, in der Zieldatei *Target View*. Die Bereiche werden durch Anfangspunkt und Länge im SVN-Delta-Format exakt definiert. Die Instruktionen speichern ihrerseits die vorzunehmenden Zeichenänderungen in dem in Abbildung 4.5 dargestellten Format. Eine Instruktion darf Daten definierter Länge aus dem Source View, dem Target View oder der New Data-Sektion in die zu ändernde Datei kopieren, wobei eine Kopie aus dem Target View nur möglich ist vor und bis zur aktuellen Position der Bearbeitung. Längen und Anfangspunkte sind im SVN-Delta-Format variabel codiert, das heißt sie nutzen das höchste Bit des jeweiligen Bytes dazu, ein weiteres zu indizieren, falls die Speichergröße nicht ausreicht.

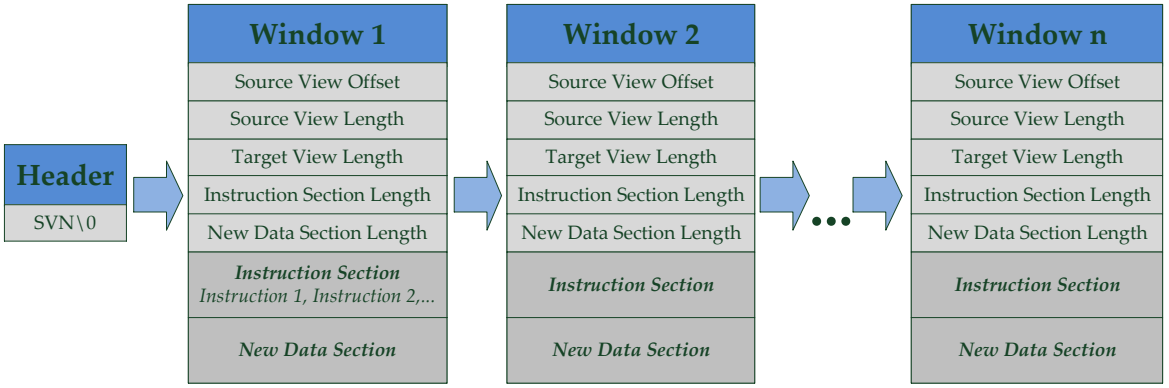

Abbildung 4.4: SVN-Delta-Format zur Speicherung von Datei-Differenzen

Zur Speicherung von Länge und Offset bleiben durch Wegfall des ersten Bits lediglich 7 Bit innerhalb eines Bytes zur Speicherung übrig, das heißt ab einem Wert von 128 wird ein zweites Byte zur Speicherung der Länge notwendig usw.

Die Nutzung des vorgestellten Subversion-Algorithmus durch JavaScript im Demonstrator realisiert den eigentlichen Zweck einer Versionsverwaltung. Er ist in der Lage, Änderungen in Dateien zu erkennen und in ein austauschbares Format zu wandeln. Die so erstellten Patches können über das Subversion-Anwendungsprotokill und die Verbs `PUT` bzw. `MERGE` in einen Arbeitsordner des Subversion-Servers geschrieben werden, wo sie auf das Projektarchiv unter Erstellung einer neuen Revision angewendet werden. Der schreibende Zugriff für URIs wird technisch definiert durch die Protokolle HTTP und WebDAV [5, 19] und mit Hilfe von sogenannten Write-Tokens realisiert. Für den vorgenommenen Schreibzugriff wird ein nutzerbasiertes Token erzeugt und der URI für andere Nutzer zum Schreiben gesperrt.

Die wesentlichen Funktionen von Subversion wurden damit in einem Demonstrator prototypisch umgesetzt. Erst die Nutzung des persistenten Webbrowser-Speichers erlaubt eine sinnvolle Anwendung eines Versionsverwaltungs-Client im Webbrowser. Darüber hinaus sind Techniken wie Subversion bereits auf eine Interaktion im Netzwerk ausgelegt, so dass das Ergebnis der technischen Machbarkeit wenig überrascht.

4.3 Bewertung im Kontext installationsfreier Client-Software

Die Entwicklung des prototypischen Demonstrators zeigt, dass Web-Anwendungen prinzipiell in der Lage sind, auch für angebotene Server-Dienste wie eine Versions-

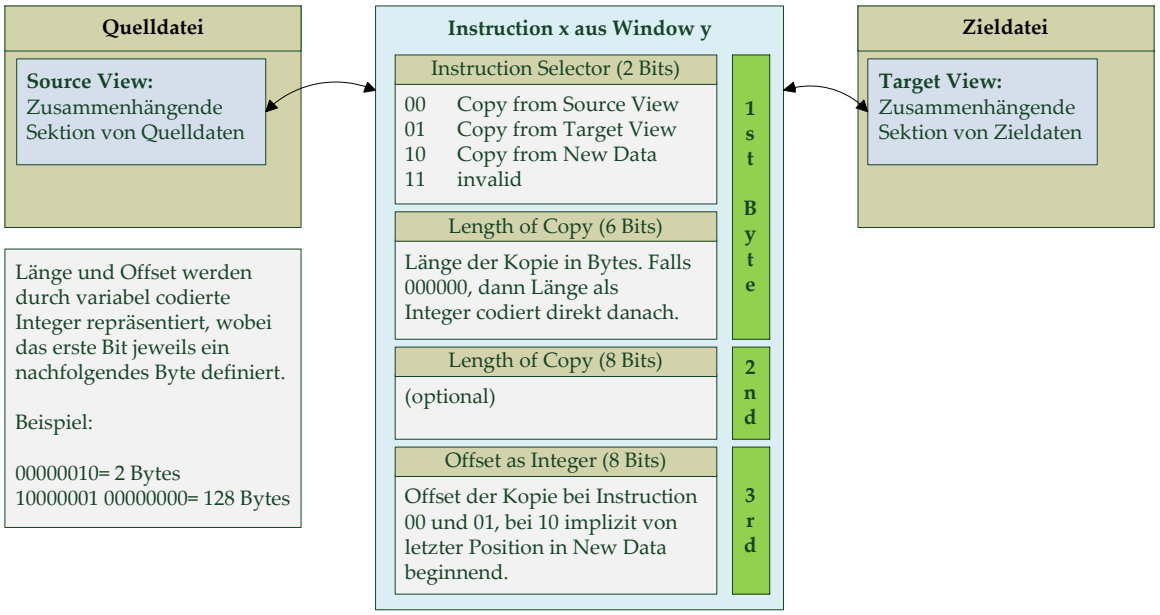

Abbildung 4.5: Transformation der Quell- zur Zieldatei mittels Instructions

verwaltung eine Client-Software bereitzustellen. Die Funktionsfähigkeit wird zwar durch das Sandbox-Prinzip und fehlende portable Mechanismen zum Datenaustausch eingeschränkt, allerdings eröffnen sich damit wiederum andere Anwendungsgebiete. Bei mehreren Internet-Anbietern können sich Anwender bereits Online-Speicherplatz reservieren, der jederzeit zur Verfügung steht und zudem abgesichert ist. Die Beschränkung von Web-Anwendungen auf der Client-Seite bezogen auf das dort vorhandene Dateisystem kann daher in Zukunft durch solche Entwicklungen auch ein irrelevanter Faktor für die Vollständigkeit einer Software werden.

4.3.1 Vergleich mit anderen Lösungen

Die typische Nutzung von Versionskontrollsystemen besteht heutzutage in dem Austausch von Daten zwischen dem Dateisystem des Betriebssystems eines Rechners und dem Repository. Zwischen den Übertragungen können auf beiden Seiten Änderungen vorgenommen werden, die zu irgendeinem Zeitpunkt wieder synchronisiert werden. Zusätzlich erstellte Metadaten wie zum Beispiel eine inhaltliche Beschreibung der Änderungen verhelfen zu einem generellen Überblick und einer vollständigen Versionsgeschichte der Daten. Für viele enthaltene Daten wie Programm-Quelltexte ist das Herunterladen in das Dateisystem essentiell, um sie dort beispielsweise übersetzen

zu können. Andere Dateien müssen hingegen von einem speziellen Programm auf dem Zielrechner bearbeitet werden, so dass auch hier die Speicherung zunächst im Dateisystem vorgenommen werden muss.

Das ist prinzipiell auch mit dem Webbrowser möglich. Allerdings schirmt die Sandbox des Webbrowsers die Web-Anwendung vollständig vom Betriebssystem ab, das heißt sie ist nicht in der Lage, selbstständig im Dateisystem des Rechners zu speichern oder Dateien von dort zu laden. Die einzige portable und auch bisher verwendete Schnittstelle sind modale Download- und Upload-Fenster, die sämtliche anderen Webbrowser-Fenster blockieren, wenn der Nutzer eine Entscheidung für den Datenaustausch treffen soll. Er muss in dem Fall Dateien einzeln hoch- oder herunterladen, die Ordnerstruktur des Projektarchivs oder der Arbeitskopie geht verloren. Es existieren unzählige andere und leicht modifizierte Ansätze zur Änderung dieses Verhaltens wie zum Beispiel Drag-&-Drop Mechanismen oder ein automatisches Herunterladen des Webbrowsers, wenn eine Datei angeboten wird. Auch HTML5 definiert Drag-&-Drop-Methoden zur Vereinfachung von Upload und Download [25]. Nichtsdestotrotz verliert die Web-Anwendung bei allen diesen Techniken das Wissen über den Speicher- oder den Herkunftsort der ausgetauschten Daten, womit ein sinnvoller mehrfacher und nachverfolgbarer Datenaustausch mit dem Dateisystem des Rechners unmöglich wird. Außerdem bietet auch erst HTML5 die Möglichkeit, eine Struktur von Ordnern und Dateien zu erkennen und zu erhalten, während momentan lediglich der Upload und Download von jeweils einer Datei spezifiziert ist und Ordner generell nicht behandelt werden.

Gegenüber Client-Software wie betriebssystemabhängiger Software oder Software auf Basis von portablen Software-Plattformen, die durch eine Installation eingerichtet wird, fehlt es installationsfreier Client-Software bisher noch an portablen und standardisierten Konzepten zur funktionellen Erweiterung. Diese wären aber notwendig, um beispielsweise Zugriff auf das Dateisystem des Rechner erlangen zu können. Ein vollständiger Funktionsumfang wird derzeit daher nur durch eine Form der Installation einer Software erreicht, auch wenn diese wie bei netzwerkgestützten Software-Plattformen eventuell minimal ausfällt. Die bereits angesprochene Entwicklung von HTML5 ermöglicht allerdings generell neue Software-Konzepte wie den Austausch von Daten über Online-Speicherdienste. Auch die Vereinheitlichung der Kommunikationsform von Servern und Clients mittels REST kann selbst bei bestehenden Einschränkungen als vorteilhafte Entwicklung angesehen werden. Unterschiedlichste Dienste werden auf diese Art und Weise über dasselbe Protokoll wie HTTP ansprechbar. Das reduziert den Implementationsaufwand für die Unterstützung neuer REST-basierter Dienste.

4.3.2 Erweiterungsmöglichkeiten

Da der kommende Standard HTML5 auch ein erweitertes API zur Behandlung von Drag-&-Drop-Ereignissen zwischen Webbrowser und Betriebssystem enthalten wird [25], ist zumindest eine Nutzung dessen denkbar. Eine Synchronisierung *vom* Dateisystem des Rechners *in* das virtuelle Dateisystem des Webbrowsers ist dann sogar mit Erhaltung der Ordner- und Dateistruktur möglich. In die andere Richtung verbietet aber nach wie vor die Sandbox eine Synchronisierung und verhindert damit den Prozess des Checkouts in das Dateisystem des Rechners. Andere Entwicklungen sind zum Zeitpunkt der Erstellung der Arbeit nicht abzusehen.

4.3.3 Machbarkeitsabschätzung

Das zentrale Thema bei der Nutzung von Webtechnologien bleibt weiterhin die Programmierung mit und an den technischen Grenzen, die von der Sandbox vorgegeben werden. Eine vollständige Integration in das Betriebssystem des Rechners ist derzeit noch nicht möglich. Die größte Hürde stellt hierbei die Trennung der Web-Anwendung vom Dateisystem des Rechners dar, welche jedoch bei installationsfreier Client-Software aufgrund der angesprochenen Sicherheitsbedenken eine unstrittige Vorsichtsmaßnahme ist. Da momentan auch keine portablen Möglichkeiten für eine sichere Funktionserweiterung wie bei Java Applets existieren, sind die Techniken JavaScript, CSS und auch das noch nicht standardisierte HTML5 beeinträchtigt in der Form der Realisierung einer Anwendung. Für die konkrete Nutzung als SimMoLib-Web-Client scheint die Nutzung von Webtechnologien allein daher derzeit keine Alternative zu dem bestehenden javabasierten Client zu sein, wenn ein Austausch mit dem Dateisystem des Rechners notwendig ist. Der Umweg über einen manuellen Upload- und Download im Webbrowser und der Verlust der Datei- und Ordnerstruktur verhindern eine sinnvolle Umsetzung, da eine Synchronisation zwischen den Daten im Webbrowser-Speicher und den Daten im Dateisystem nicht stattfinden kann. Nichtsdestotrotz bietet der Demonstrator einen Einblick in die Möglichkeiten von installationsfreier Client-Software. Für eine dokumentenbasierte Bearbeitung im Webbrowser sind alle notwendigen Techniken vorhanden und umsetzbar. Kollaborative Plattformen sind auf diese Weise möglich und bieten durch die Versionsverwaltung gleichzeitig eine Nachverfolgbarkeit und Versionsgeschichte.

4.3.4 Anwendungsgebiet

Die kollaborative Arbeit an Dateien und eine zusätzliche Versionierung im Webbrowser wird bereits von Diensten im Internet umgesetzt und angeboten. Sinnvoll ist das bei Dokumenten von Textverarbeitungsprogrammen oder Tabellenkalkulationsprogrammen, die gleichzeitig im Webbrowser dargestellt und bearbeitet werden können. Zudem entfällt für die Nutzergruppe eine mühsame Synchronisierung oder gar das Warten auf die Möglichkeit zur Bearbeitung eines Dokuments, da die Versionsverwaltung das Zusammenfügen der Änderungen in den einzelnen Arbeitskopien wie in Abschnitt 2.7.4 beschrieben übernimmt. Sollten die Dienste von Online-Speicherplatz im Netzwerk oder in der Cloud weiter ausgebaut werden, besteht außerdem die Möglichkeit, dass sich der Webbrowser und das Betriebssystem diesen Speicherplatz teilen. Auf diese Weise würde der Webbrowser das eigentliche Dateisystem des Rechners nicht beschädigen können, wäre allerdings dazu in der Lage, dem Rechner eine vollständige Dateien- und Ordnerstruktur wie den Inhalt eine Arbeitskopie direkt zur Verfügung zu stellen.

Die für das Beispiel vorgestellten und verwendeten Techniken sind in den aktuellen Versionen der Webbrowser Chrome, Firefox und Opera verfügbar, auch wenn der HTML5-Standard noch nicht abgeschlossen spezifiziert wurde. Der aktuelle Internet Explorer 9 unterstützt das FileSystem-API nicht. Die fehlende Standardisierung verursacht in jedem Fall Probleme bei der portablen Programmierung, denn es werden durch die rasche Entwicklung einige Objekte in JavaScript durch Präfixe wie `webkit-` oder `mozilla-` auf einen einzigen Webbrowser beziehungsweise auf seine Darstellungsumgebung festgelegt. Zum Teil unterstützen zwar Chrome und Firefox dann die gleichen Funktionen, der Programmierer muss diese aber webbrowserabhängig und unter Umständen mit anderen Parametern aufrufen. Auch Änderungen zwischen Webbrowser-Versionen sind möglich, so dass eine zusätzliche Wartung notwendig werden kann. Eine explizite Aussage über die notwendige Webbrowser-Plattform scheint daher nur im konkreten Anwendungsfall sinnvoll zu sein, auch wenn Firefox und Chrome bisher bei der Unterstützung von HTML5 in jedem Fall vor dem Internet Explorer liegen, da Microsoft absichtlich auf die Unterstützung nicht veröffentlichter Standards im Internet Explorer verzichtet hat [23].

5 Zusammenfassung und Ausblick

Die Motivation zur Untersuchung von installationsfreier Client-Software ergab sich aus einer konkreten Anwendung des Deutschen Zentrums für Luft- und Raumfahrt. Die Bereitstellung einer möglichst einfach zugreifbaren Bibliothek für Simulationsmodelle, deren Änderungen und Versionen erhalten bleiben sollen, wird dort im Rahmen des Projekts SimMoLib untersucht. Mangels vorliegender Definitionen wurde der Begriff *installationsfreie Client-Software* innerhalb des Gefüges von unterschiedlichen Software-Arten eingeordnet. Betriebssystemabhängig installierte Software bietet nach wie vor die größten Möglichkeiten zur Anpassung und die mit Abstand höchste Funktionalität. Sie kann in der Regel auf die Gesamtheit der Betriebssystemfunktionen und auf sämtliche Hardware mittels Treibern zurückgreifen. Ebenso wenig beschränkt sind prinzipiell die technischen Möglichkeiten und Rechte zur Nutzung von Sekundärspeicher, Rechenzeit und der Netzwerkkommunikation.

Den Mangel an Portabilität von betriebssystemabhängiger Software beheben portable Software-Plattformen wie Java und .NET auf Kosten einer zusätzlichen Installation und regelmäßiger Aktualisierung der jeweiligen Plattform. Dennoch sind diese mittlerweile sehr weit verbreitet. Der Nachteil einer ineffizienten Interpretation eines portablen Zwischencodes innerhalb einer virtuellen Maschine wird durch Techniken wie Just-in-Time-Compiler und die hardwarenahe Programmierung der virtuellen Maschine fast aufgehoben. Zudem ist die Nutzung solcher Plattformen in ähnlicher oder abgewandelter Form in Kleinstgeräten wie Smartphones ein Zeichnen für ihre prinzipielle Leistungsfähigkeit. Netzwerkgestützte Software-Plattformen bieten ähnliche Funktionen, aber zusätzlich eine Reduktion des Wartungsaufwands der durch sie ausgeführten Programme, da sie über Netzwerkverbindungen in einer Client-Server-Infrastruktur automatisch aktualisiert werden.

Der Kreis schließt sich mit installationsfreier Client-Software, deren Bezeichnung sich vollständig auf Web-Anwendungen zurückführen ließ, die auf dem Client-Rechner im Webbrowser ablaufen. Sie werden einerseits sofort ausgeführt und andererseits wohnt ihnen die Eigenschaft inne, sich als Client-Software mit Server-Diensten wie einem Web-Server oder einem Versionsverwaltungs-Server zu verbinden. Maßgeblich hierfür ist die vorgestellte Programmiersprache JavaScript, welche von den Webbrowsern interpretiert

wird. In den meisten Fällen dienen Web-Anwendungen als Client-Software für Inhalte eines Web-Servers und zeigen diese an. Dennoch wurden auch Techniken wie CouchDB erwähnt, die eine Datenbank-Funktion für Web-Anwendungen realisiert. Die Art der Kommunikation zwischen Client und Server erfolgt dabei sinnvollerweise durch das vorgestellte REST-Prinzip.

Die Ausführung und die Fähigkeiten von installationsfreier Client-Software im Webbrowser werden durch das implementierte Sandbox-Prinzip eingeschränkt. Gegenläufig dazu führen neue Techniken wie HTML5 zu umfangreicheren Programmiermöglichkeiten und lösen damit bereits Sandbox-Einschränkungen auf. Zusätzlich beschleunigen Netzwerke wie das Internet und die damit verbundene Client-Server-Infrastruktur die Entwicklung von neuen Anwendungsbereichen erheblich. Eine dauerhaft bestehende Netzwerkverbindung bietet dem Client die Möglichkeit, viele seiner Ressourcen, Anwendungen und Dienste auf Server auslagern zu können. Dort stehen die Ressourcen bedarfsgerecht jederzeit zur Verfügung. Das *Outsourcing in die Cloud* birgt jedoch auch Gefahren wie den Kontrollverlust über die eigenen Daten. Das Recht auf informationelle Selbstbestimmung scheint bei solchen Anwendungen im Internet kaum durchsetzbar zu sein, da die meisten Anbieter von Web-Anwendungen transparent für den Nutzer personenbezogene Daten sammeln, um damit beispielsweise Werbung platzieren zu können. Ebenso könnten ausgelagerte Daten wie die von Versionsverwaltungen einer Fremdnutzung unterliegen, die nicht nachvollzogen werden kann. Zudem verursacht die Nutzung der Cloud eine zusätzliche Abhängigkeit des Nutzers, denn wo Daten und Software zusammen aufbewahrt werden, kann der Anbieter die Richtlinien zur Nutzung oder die Preise hierfür fast unkontrolliert diktieren. Es bleibt daher abzuwarten, ob klassisch installierte und netzwerkunabhängige Software auf Rechnern durch Web-Anwendungen letztendlich ganz verdrängt wird.

Gerade in einer Netzwerkstruktur mit vielen Teilnehmern, die potentiell miteinander kommunizieren und Daten austauschen, sind Sicherheitsmechanismen daher noch wichtiger als bei isolierten Rechnern. Gängige Praxis zum Datenschutz bei Firmen ist zum Beispiel die Erstellung von privaten lokalen Netzwerken oder das Filtern von Informationen, die aus einem sicherheitskritischen Netzwerk wie dem Internet abgerufen werden dürfen. In einem vertrauenswürdigen Netzwerk hingegen könnte installationsfreie Client-Software die Verteilung und Nutzung von bestimmten Diensten wesentlich vereinfachen, da weder eine Installation noch eine client-basierte Aktualisierung oder Wartung notwendig sind. Die meisten verfügbaren Desktop-Betriebssysteme enthalten außerdem bereits einen Webbrowser, also eine passende Plattform zur Interpretation. Die minimalen Hürden bei der Bedienung von Web-Anwendungen könnten den Anwendungsbereich von Versionsverwaltungssystemen außerdem dahingehend erwei-

tern, dass die Versionierung von Daten nicht nur in der Programmierung stattfindet. Jegliche in digitaler Form vorhandenen Daten, die überschrieben oder verändert werden, verlieren ohne ein Versionsverwaltungssystem ihre Entstehungsgeschichte und möglicherweise wichtige Informationen.

Am Beispiel der Versionsverwaltung Subversion wurden die technischen Möglichkeiten von installationsfreier Client-Software und Algorithmen aufgezeigt, die erst durch die neuartigen Entwicklungen realisierbar sind. Dabei sind effiziente Just-in-Time-Compiler für JavaScript zur Verbesserung des Laufzeitverhaltens ebenso wichtig wie nichtflüchtiger Sekundärspeicher durch das FileSystem-API und die Netzwerkkommunikation mittels REST, was auch und gerade von der Server-Seite durch Subversion unterstützt wird. Techniken zur einfachen Änderung der Oberfläche einer Web-Anwendung und eine benutzerfreundliche Interaktion durch im Hintergrund asynchron ablaufende Funktionen sind bereits länger im Einsatz und ebenso wichtig für eine sinnvoll zu bedienende Web-Anwendung.

5.1 Aktuelle Entwicklungen

Die aktuellen Entwicklungen innerhalb der möglichen Client-Server-Strukturen ist beinahe unüberschaubar. Neue Entwicklungen ergeben sich oftmals aus neuen Anforderungen für Web-Anwendungen, die dann Änderungen von Websites und Web-Anwendungen nach sich ziehen. Der Begriff *Living Standards*, mit welchem die WHATWG, eine der aktivsten Entwicklergruppen für Webtechnologien, die dynamischen und schnelllebigen Standards bezeichnet [24], ist prägend für diese schnellen Änderungen. Einerseits wird installationsfreie Client-Software durch Entwürfe für HTML5 ständig um neue Fähigkeiten erweitert, andererseits verlagern Software-Hersteller ihre Angebote teilweise ebenso schnell in das Netzwerk. Dennoch oder gerade deswegen werden jedoch Speichermöglichkeiten auf dem Client benötigt, da sich die Verzögerungen durch die Signalübertragung im Netzwerk immer noch stark auf die Nutzbarkeit einer Software auswirken können. Um trotzdem interaktive flüssig ablaufende Anwendungen anbieten zu können, werden beispielsweise statische und umfangreiche Daten auf dem Client zwischengespeichert, damit vom Web-Server beim erneuten Zugriff lediglich eine möglichst minimale Aktualisierung der Daten heruntergeladen werden muss. Trotzdem sind die Latenzen innerhalb großer Netzwerke wie dem Internet bisher noch ein großes Problem für die Benutzerfreundlichkeit. Auch Online-Speicher-Dienste in der Cloud wie Microsoft SkyDrive oder Amazon S3 [CKS09], die als Ersatz für lokalen Speicher auf dem Client dienen können, sind davon betroffen. Der Erfolg des Cloud Compu-

ting, also der Bereitstellung aller möglichen Dienste und Software aus dem Netzwerk, wird maßgeblich von der Weiterentwicklung der technischen Übertragungssysteme und Anbindungen an das Netzwerk abhängen. Während in anderen Ländern deshalb bereits verbreitet Haushalte mit Glasfaser-Leitungen an den Provider angeschlossen werden, sind in Deutschland nach wie vor Kupferleitungen des Kabelfernsehens oder des Telefons die Hauptübertragungsmedien. Eine optische Signalübertragung gewährt derzeit jedoch ein Minimum an Latenzen.

Die vorgestellten Techniken für installationsfreie Client-Software namens Flash, Silverlight, Java FX/Applets und die Trinität aus HTML, CSS und JavaScript sind derzeit etablierte Technologien. Ihr Anwendungsbereich unterscheidet sich durch viele Überschneidungen teilweise noch, daher ist ihr Nebeneinander in Web-Anwendungen alltäglich. Flash ist beispielsweise nach wie vor führend im Markt der Internet-Werbung, da die darzustellenden Inhalte einfach und umfangreich animiert werden können. Java-Anwendungen kommen oft dort zum Einsatz, wo in jedem Fall viel Interaktivität mit dem Betriebssystem und zusätzliche Funktionalität notwendig sind und sich der Nutzer mit der etwas längeren Startzeit durch die Laufzeitumgebung arrangieren kann. Die Standardisierung und rapide Entwicklung im Bereich HTML5 führt allerdings zu einer Konkurrenzsituation zwischen den Techniken. Manche Firmen unterstützen bereits offiziell eine der Techniken nicht mehr, da diese jeweils keine essentiell notwendige Funktionalität für Web-Anwendungen bereitstellen, wenn einmal von einem minimalen HTML-Gerüst abgesehen wird, das die jeweilige Technik initialisiert. Es ist daher nicht auszuschließen, dass sich die Zahl der Techniken in Zukunft reduziert. Eine Aussage zur Zukunftssicherheit scheint wegen der umfassenden Entwicklung von HTML5 derzeit nur bei den nativen Techniken der Webbrowser, also HTML, CSS und JavaScript sinnvoll zu sein.

Die durch die Sandbox definierte funktionale Grenze von Web-Anwendungen ist allerdings keine verlässliche Größe. Mit WebGL wurde eine Möglichkeit für Web-Anwendungen geschaffen, die hardwarenahe und verbreitete Implementationssprache von Grafikkarten namens OpenGL für eine beschleunigte Darstellung von grafischen Inhalten zu nutzen. Das ist prinzipiell sinnvoll bei aufwändigen grafischen Daten, die zudem direkt beim Anschauen vergrößert, verkleinert oder generell verändert werden sollen wie zum Beispiel satellitengestützten Landschaftsaufnahmen. Grafikkarten sind bei solchen Aufgaben wesentlich effizienter und damit schneller, da sie im Gegensatz zu Hauptprozessoren eines Rechners mit vielen und dafür weniger komplexen Rechenkernen massiv parallel arbeiten. Grafische Daten lassen sich aufgrund ihrer vorhersagbaren ähnlichen Struktur sehr gut parallel und mit relativ primitiven Befehlen verarbeiten. Dem gegenüber stehen die Bemühungen, die massive Parallelität der Rechenkerne von

Grafikkarten auch für Programme zu nutzen, die normalerweise vom Hauptprozessor ausgeführt werden. Das beschleunigt beispielsweise die Erzeugung von kryptografischen Hash-Funktionen, mit denen Dateien auf ihre Integrität überprüft werden können. Beide Anwendungsbereiche zusammen führen bei Implementationsfehlern mitunter zu ungewollten Effekten, denn WebGL ist natürlich nur dafür gedacht, grafischen Inhalt zu beschleunigen und nicht Programme anstelle des Hauptprozessors auszuführen. Der Mangel eines definierten Sandbox-Verhaltens von Webbrowsern und die teilweise Auflösung durch Flash oder durch HTML5 ist für die Sicherheit von Rechnern zumindest bedenklich. Ähnlich problematisch verhält es sich bereits seit längerer Zeit mit ActiveX als Funktionserweiterung für den Internet Explorer. Es bleibt daher zu hoffen, dass die zunehmenden Fähigkeiten von Web-Anwendungen ebenso zu einer klaren technischen Definition der Sandbox führen beziehungsweise einheitliche Konzepte für Software umgesetzt werden, die die Sandbox verlassen muss, um ihre Funktion zu erfüllen.

Literaturverzeichnis

[AFG+10] ARMBRUST, Michael ; FOX, Armando ; GRIFFITH, Rean ; JOSEPH, Anthony D. ; KATZ, Randy ; KONWINSKI, Andy ; LEE, Gunho ; PATTERSON, David ; RABKIN, Ariel ; STOICA, Ion ; ZAHARIA, Matei: A view of cloud computing. In: *Commun. ACM* 53 (2010), April, S. 50–58. – ISSN 0001–0782

[AS09] ALWIS, Brian de ; SILLITO, Jonathan: Why are software projects moving from centralized to decentralized version control systems? In: *Proceedings of the 2009 ICSE Workshop on Cooperative and Human Aspects on Software Engineering*. Washington, DC, USA : IEEE Computer Society, 2009 (CHASE '09), S. 36–39

[Bau10] BAUN, Christian: *Cloud computing : web-basierte dynamische IT-Services*. Berlin [u.a.] : Berlin [u.a.] : Springer, 2010

[Ber96] BERSON, Alex: *Client/server architecture (2nd ed.)*. New York, NY, USA : McGraw-Hill, Inc., 1996. – ISBN 0–07–005664–1

[Bux08] BUXMANN, Peter: Software as a Service. In: *WIRTSCHAFTSINFORMATIK, 2008, Vol.50(6), p.500-503* 50 (2008), Nr. 6, S. 500

[BW09] BIANCUZZI, Federico ; WARDEN, Shane: *Masterminds of Programming: Conversations with the Creators of Major Programming Languages*. O'Reilly Media, 2009. – 494 S. – ISBN 0–596–51517–0

[CDH07] CHAMBERS, Mike ; DURA, Daniel ; HOYT, Kevin: *Adobe integrated runtime (air) for javascript developers pocket guide*. First. O'Reilly, 2007. – ISBN 9780596515195

[CKS09] CACHIN, Christian ; KEIDAR, Idit ; SHRAER, Alexander: Trusting the cloud. In: *SIGACT News* 40 (2009), Juni, Nr. 2, S. 81–86. – ISSN 0163–5700

[Fie00] FIELDING, Roy: *Architectural Styles and the Design of Network-based Software Architectures*, University of California, Irvine, Diss., 2000

[GWM01] GOLL, Joachim ; WEISS, Cornelia ; MÜLLER, Frank: *Java als erste Programmiersprache*. B. G. Teubner, Stuttgart/Leipzig/Wiesbaden, 2001

[HM76] HUNT, J. W. ; MACILROY, M. D.: *An algorithm for differential file comparison*. N.J., USA : Bell Laboratories, Murray Hill, N.J., 1976

[Kay96] KAY, Alan C.: History of programming languages—II. New York, NY, USA : ACM, 1996. – ISBN 0–201–89502–1, Kapitel The early history of Smalltalk

[Ker10] KERL, Christian: *Realisierung einer verteilten Bibliothek für wiederverwendbare dynamische Simulationsmodelle*, Duale Hochschule Baden-Württemberg, Bachelorarbeit, 2010

[Kof07] KOFLER, Michael: *Linux*. München, Deutschland [u.a.] : Addison-Wesley, Pearson Education, 2007

[KP88] KRASNER, Glenn E. ; POPE, Stephen T.: A cookbook for using the model-view controller user interface paradigm in Smalltalk-80. In: *J. Object Oriented Program.* 1 (1988), August, Nr. 3

[KR78] KERNIGHAN, Brian W. ; RITCHIE, Dennis M.: *The C Programming Language*. Prentice Hall International, Englewood Cliffs, New Jersey, 1978. – ISBN 0–131–10362–8

[Kru10] KRUG, Steve: *Web Usability*. Addison-Wesley, München, 2010. – 183 S.

[LAD+12] LÜDTKE, Daniel ; ARDAENS, Jean-Sébastien ; DESHMUKH, Meenakshi ; LOPEZ, Rosa P. ; BRAUKHANE, Andy ; THEIL, Stephan ; GERNDT, Andreas: Collaborative Development and Cataloging of Simulation and Calculation Models for Space Systems [submitted]. In: *3rd IEEE Track on Collaborative Modeling and Simulation*. Washington, DC, USA : IEEE Computer Society, 2012 (21st IEEE International Conference on Collaboration Technologies and Infrastructures, June 25-27, Toulouse (France))

[LB99] LIE, Håkon W. ; BOS, Bert: *Cascading Style Sheets, designing for the Web (2nd edition)*. München, Deutschland [u.a.] : Addison Wesley, 1999. – ISBN 0–201–59625–3

[Lev66] LEVENSHTEIN, V. I.: Binary codes capable of correcting deletions, insertions and reversals. In: *Soviet Physics Doklady* 10 (1966), S. 707–710

[Mar01] MARINILLI, Mauro: *Java Deployment with Jnlp and Webstart*. Sams Publishing, Pearson Schweiz AG, Zug, Schweiz, 2001. – ISBN 0–672–32182–3

[Men02] MENS, Tom: A State-of-the-Art Survey on Software Merging. In: *IEEE Transactions on Software Engineering* 28 (2002), May, S. 449–462. – ISSN 0098–5589

[Moc09] MOCKUS, Audris: Amassing and indexing a large sample of version control systems: Towards the census of public source code history. In: *Proceedings of the 2009 6th IEEE International Working Conference on Mining Software Repositories*. Washington, DC, USA : IEEE Computer Society, 2009 (MSR '09). – ISBN 978–1–4244–3493–0

[MS00] MESSERSCHMITT, David G. ; SZYPERSKI, Clemens: Industrial and Economic Properties of Software - Technology, Processes, and Value / University of California. Berkeley, California, USA, 2000. – Forschungsbericht

[Mye86] MYERS, Eugene: An O(ND) difference algorithm and its variations. In: *Algorithmica* 1 (1986), 251-266. `http://dx.doi.org/10.1007/BF01840446`. – ISSN 0178–4617. – 10.1007/BF01840446

[Neu93] NEUMANN, John von: First Draft of a Report on the EDVAC. In: *IEEE Ann. Hist. Comput.* 15 (1993), Oktober, Nr. 4

[Nic09] NICOLA, Carlo U.: Einblick in die Dalvik Virtual Machine. In: *IMVS Fokus Report* Volume 3, Issue 1: 5-12, 2009 (2009)

[Rin10] RINGELS, Holger: *Smartclients mit Eclipse RCP: Architektur und Konzeption von Enterprise-Anwendungen*. entwickler.press Verlag, Software & Support Media GmbH, Frankfurt am Main, 2010. – ISBN 3–868–02049–7

[Sch01] SCHILDT, Herbert: *C Ent-Packt*. MITP-Verlag, Bonn, 2001. – ISBN 3–826–60732–5

[SFR04] STEVENS, W.R. ; FENNER, B. ; RUDOFF, A.M.: *UNIX Network Programming: The sockets networking API*. Addison-Wesley, 2004 (Addison-Wesley professional computing series). – ISBN 9780131411555

[SG98] SILBERSCHATZ, Abraham ; GALVIN, Peter B.: *Operating System Concepts*. Reading, Massachusetts [u.a.] : Addison-Wesley, Addison Wesley Longman, Inc., 1998

[Tur99] TURAU, Volker: Techniken zur Realisierung Web-basierter Anwendungen. In: *Informatik-Spektrum* Volume 22: 3-12, 1999 (1999)

[Ull09] ULLENBOOM, Christian: *Java ist auch eine Insel (8. Auflage)*. Bonn, Deutschland : Galileo Computing, Galileo Press GmbH, 2009. – ISBN 3–83–621371–0

[Wil56] WILKES, M. V.: *Automatic Digital Computers*. John Wiley & Sons, 1956

Internetquellenverzeichnis

[1] ADOBE SYSTEMS INCORPORATED: *ACROBAT.COM*. Website. http://www.acrobat.com. Version: 2011, Abruf: 15.03.2012

[2] APACHE FOUNDATION: Kein Titel *(SVN-Delta Format-Beschreibung)*. Website. http://svn.apache.org/repos/asf/subversion/trunk/notes/svndiff. Version: 2011, Abruf: 09.03.2012

[3] APPLE INC.: *SunSpider JavaScript Benchmark*. Website. http://www.webkit.org/perf/sunspider/sunspider.html. Version: 2011, Abruf: 24.02.2012

[4] AWIO WEB SERVICES LLC, INTERNETDIENSTLEISTER: *(W3Counter Report for) January 2012*. Website. http://www.w3counter.com/globalstats.php?year=2012&month=01. Version: 2012, Abruf: 07.02.2012

[5] BERNERS-LEE, Tim ; FIELDING, Roy ; FRYSTYK, H.: *RFC 1945: Hypertext Transfer Protocol – HTTP/1.0*. Website. http://www.ietf.org/rfc/rfc1945.txt. Version: 1996, Abruf: 08.01.2012

[6] BIDELMAN, Eric: *Exploring the FileSystem APIs*. Website. http://www.html5rocks.com/en/tutorials/file/filesystem/. Version: 2012, Abruf: 18.02.2012

[7] BORNSTEIN, Dan: *Presentation of Dalvik VM Internals*. Presentation at Google I/O Developer Conference, 2008. http://www.imamu.edu.sa/dcontent/IT_Topics/java/2008-05-29-presentation-of-dalvik-vm-internals.pdf. Version: 2008, Abruf: 11.12.2011

[8] BROCKHAUS ENZYKLOPÄDIE ONLINE, BIBLIOGRAPHISCHES INSTITUT & F. A. BROCKHAUS AG: *Installation*. Website. http://www.brockhaus-enzyklopaedie.de/be21_article.php#4. Version: 2011, Abruf: 11.01.2012

[9] BROWN, Michael: *iPXE - open source boot firmware*. Website. http://ipxe.org. Version: 2011, Abruf: 25.12.2011

[10] BROWSERRANK.COM, INTERNETDIENSTLEISTER, STATISKEN VON HTTP://WWW.STATCOUNTER.COM: *Most Popular Browser By Country*. Website. http://www.browserrank.com/. Version: 2012, Abruf: 04.03.2012

[11] CERF, Vinton G.: *RFC 20: ASCII format for Network Interchange*. Website. http://tools.ietf.org/html/rfc20. Version: 1969, Abruf: 02.02.2012

[12] CHACON, Scott: *The Git Community Book*. Website. http://book.git-scm.com. Version: 2012, Abruf: 16.02.2012

[13] CLEMM, G. ; AMSDEN, J. ; ELLISON, T. ; KALER, C. ; WHITEHEAD, J.: *RFC 3253: Versioning Extensions to WebDAV (Web Distributed Authoring and Versioning)*. Website. http://www.ietf.org/rfc/rfc3253.txt. Version: 2002, Abruf: 10.01.2012

[14] COLLINS-SUSSMAN, Ben ; FITZPATRICK, Brian W. ; PILATO, C. M.: *Version Control with Subversion*. Website. http://svnbook.spears.at/en/1.5/svn-book.html. Version: 2008, Abruf: 14.02.2012

[15] COMPUTERWORLD INC., NACHRICHTENDIENSTLEISTER IM INTERNET: *Google fixes 9 bugs in Chrome, including sandbox-escape flaw*. Website. http://www.computerworld.com/s/article/9208279/Google_fixes_9_bugs_in_Chrome_including_sandbox_escape_flaw. Version: 2011, Abruf: 21.01.2011

[16] DREAMINGWELL.COM, INTERNET SERVICES COMPANY, USA: *Rich Internet Application Statistics*. Website. http://www.riastats.com/#. Version: 2011, Abruf: 06.10.2011

[17] DREPPER, Ulrich: *How To Write Shared Libraries*. Website. http://nuclear.dnsalias.com/tmp/dsohowto.pdf. Version: 2010, Abruf: 30.11.2011

[18] ECMA INTERNATIONAL, GENF, SCHWEIZ: *ECMAScript Language Specification (ECMA-262, Version 5.1, Juni 2011)*. Online-Ressource. http://www.ecma-international.org/publications/files/ECMA-ST/Ecma-262.pdf. Version: 2011, Abruf: 10.02.2012

[19] GOLAND, Y. ; WHITEHEAD, E. ; FAIZI, A. ; CARTER, S. ; JENSEN, D.: *RFC 2518: HTTP Extensions for Distributed Authoring – WEBDAV*. Website. http://www.ietf.org/rfc/rfc2518.txt. Version: 1999, Abruf: 08.01.2012

[20] GOOGLE, INC.: *V8 JavaScript Engine*. Website. http://code.google.com/p/v8/. Version: 2011, Abruf: 29.01.2012

[21] GRIBBLE, Cheryl: *History of the Web - Beginning at CERN*. Website. http://www.hitmill.com/internet/web_history.html. Version: 2009, Abruf: 20.02.2012

[22] GUTHRIE, Scott: *.NET Framework Library Source Code now available*. Website. http://weblogs.asp.net/scottgu/archive/2008/01/16/net-framework-library-source-code-now-available.aspx. Version: 2008, Abruf: 20.02.2012

[23] HACHAMOVITCH, Dean: *HTML5, Site-Ready and Experimental*. Website. http://blogs.msdn.com/b/ie/archive/2010/12/21/html5-site-ready-and-experimental.aspx. Version: 2010, Abruf: 10.03.2012

[24] HICKSON, Ian: *FAQ*. Website. http://wiki.whatwg.org/wiki/FAQ#What_does_.22Living_Standard.22_mean.3F, Abruf: 20.02.2012

[25] HICKSON, Ian: *A vocabulary and associated APIs for HTML and XHTML*. Website. http://dev.w3.org/html5/spec/Overview.html. Version: 2012, Abruf: 16.02.2012

[26] INFRATEST GMBH, INITIATIVE D21 E.V.: *(N)ONLINER Atlas 2011*. Online-Ressource. http://www.initiatived21.de/wp-content/uploads/2011/07/NOnliner2011.pdf. Version: 2011, Abruf: 20.02.2012

[27] INTEL CORPORATION: *Preboot Execution Environment (PXE) Specification, Version 2.1*. Website. http://www.pix.net/software/pxeboot/archive/pxespec.pdf. Version: 1999, Abruf: 11.12.2011

[28] JOSEFSSON, S.: *RFC 4648: The Base16, Base32, and Base64 Data Encodings*. Website. http://wiki.tools.ietf.org/html/rfc4648. Version: 2006, Abruf: 09.02.2012

[29] KESTEREN, Anne van: *The XMLHttpRequest Object*. Website. http://www.w3.org/TR/2006/WD-XMLHttpRequest-20060619/. Version: 2006, Abruf: 01.03.2012

[30] KESTEREN, Anne van: *Cross-Origin Resource Sharing*. Website. http://dvcs.w3.org/hg/cors/raw-file/tip/Overview.html. Version: 2012, Abruf: 09.03.2012

[31] KESTEREN, Anne van: *XMLHttpRequest Level 2*. Website. http://www.w3.org/TR/XMLHttpRequest/. Version: 2012, Abruf: 01.03.2012

[32] KRANTZ, Peter: *US government web sites standards usage*. Website. http://www.standards-schmandards.com/exhibits/govstandards/us/. Version: 2005, Abruf: 21.12.2011

[33] MEHTA, Nikunj ; SICKING, Jonas ; GRAFF, Eliot ; POPESCU, Andrei ; ORLOW, Jeremy: *Indexed Database API*. Website. http://www.w3.org/TR/IndexedDB/. Version: 2011, Abruf: 16.02.2012

[34] MICROSOFT CORPORATION: *Signing and Checking Code with Authenticode*. Website. http://msdn.microsoft.com/en-us/library/ie/ms537364(v=vs.85).aspx. Version: 2011, Abruf: 03.02.2012

[35] MICROSOFT CORPORATION: *Office 365 für Selbständige und kleine Unternehmen*. Website. http://office365.microsoft.com. Version: 2012, Abruf: 15.03.2012

[36] MOZILLA FOUNDATION: *JavaScript:TraceMonkey*. Website. https://wiki.mozilla.org/JavaScript:TraceMonkey. Version: 2010, Abruf: 29.01.2012

[37] MSDN, MICROSOFT DEVELOPER NETWORK: *.NET Framework 4.5 Developer Preview*. Website. http://msdn.microsoft.com/en-us/netframework/hh403373. Version: 2012, Abruf: 04.03.2012

[38] MSDN, MICROSOFT DEVELOPER NETWORK (DEUTSCHLAND): *ClickOnce-Sicherheit und Bereitstellung*. Website. http://msdn.microsoft.com/de-de/library/t71a733d.aspx. Version: 2012, Abruf: 04.03.2012

[39] MSDN, MICROSOFT DEVELOPER NETWORK (DEUTSCHLAND): *.NET Framework-Versionen und -Abhängigkeiten*. Website. http://msdn.microsoft.com/de-de/library/bb822049.aspx. Version: 2012, Abruf: 04.03.2012

[40] OLIVER DIEDRICH, Dr.: *Open-Source-Java*. Website. http://heise.de/-222009. Version: 2010, Abruf: 20.02.2012

[41] PILGRIM, Mark: *Dive into Python*. Online-Ressource. http://diveintopython.org/. Version: 2004, Abruf: 20.02.2012

[42] RED HAT, INC.: *Signing Packages*. Website. http://www.rpm.org/max-rpm/s1-rpm-pgp-signing-packages.html. Version: 2012, Abruf: 03.02.2012

[43] REKHTER, Yakov ; MOSKOWITZ, Robert ; KARRENBERG, Daniel ; GROOT, Geert J. ; LEAR, Eliot: *RFC 1918: Address Allocation for Private Internets*. Website. http://tools.ietf.org/html/rfc1918. Version: 1996, Abruf: 02.01.2012

[44] RESIG, John: *JavaScript Performance Rundown*. Website. http://ejohn.org/blog/javascript-performance-rundown/. Version: 2008, Abruf: 30.01.2012

[45] RUDERMAN, Jesse: *Signed Scripts in Mozilla*. Website. http://www.mozilla.org/projects/security/components/signed-scripts.html. Version: 2012, Abruf: 04.03.2012

[46] RÖWEKAMP, Lars: *Open-Source-Java*. Website. http://heise.de/-1354571. Version: 2011, Abruf: 05.03.2012

[47] SEIDER, Doreen: *Remote Component Environment (RCE)*. Website. http://www.dlr.de/sc/desktopdefault.aspx/tabid-5625/9170_read-17513/. Version: 2012, Abruf: 19.02.2012

[48] SHINGLEDECKER, Robert: *Core Concepts*. Website. http://distro.ibiblio.org/tinycorelinux/concepts.html. Version: 2010, Abruf: 25.11.2011

[49] SRISURESH, P. ; NETWORKS, Jasmine ; EGEVANG, K.: *RFC 3022: Traditional IP Network Address Translator (Traditional NAT)*. Website. http://tools.ietf.org/html/rfc3022. Version: 2001, Abruf: 23.01.2012

[50] STATCOUNTER, INTERNETDIENSTLEISTER STATISTA: *Marktanteile der führenden Browser an der Internetnutzung weltweit von Januar 2009 bis Januar 2012*. Website. http://de.statista.com/statistik/daten/studie/157944/umfrage/marktanteile-der-browser-bei-der-internetnutzung-weltweit-seit-2009/. Version: 2012, Abruf: 10.02.2012

[51] SÖRLIN, Johan: *JavaScript WYSIWYG Editor*. Englisch. http://www.tinymce.com. Version: 2012, Abruf: 20.03.2012

[52] THE JQUERY PROJECT: *jQuery is a new kind of JavaScript Library.* Website. http://jquery.com. Version: 2010, Abruf: 25.01.2012

[53] WEBHOSTINGREPORT.COM, INTERNETDIENSTLEISTER: *The History Of FireFox*. Website. http://www.webhostingreport.com/learn/firefox.html. Version: 2012, Abruf: 20.02.2012

[54] WHEELER, David A.: *Comments on Open Source Software / Free Software (OSS/FS) Software Configuration Management (SCM) / Revision-Control Systems*. Website. http://www.dwheeler.com/essays/scm.html. Version: 2005, Abruf: 01.02.2012

[55] WORLD WIDE WEB CONSORTIUM (W3C), INTERNATIONAL COMMUNITY OF MEMBER ORGANIZATIONS FOR WEB STANDARDS, USA: *HTML 4.01 Specification*. Website. `http://www.w3.org/TR/html401/`. Version: 1999, Abruf: 06.12.2011

[56] XAMARIN, SOFTWARE DEVELOPMENT COMPANY, USA: *FAQ: General*. Website. `http://www.mono-project.com/FAQ:_General#Mono_and_Microsoft`. Version: 2012, Abruf: 04.03.2012

Hiermit erkläre ich an Eides statt, dass ich die vorliegende Diplomarbeit selbständig verfasst und keine anderen als die angegebenen Quellen und Hilfsmittel verwendet habe. Die Arbeit wurde in gleicher oder ähnlicher Form keiner anderen Prüfungsbehörde zur Erlangung eines akademischen Grades vorgelegt.

Berlin, den 1. Juni 2012

_____

Robert Schumann