Online-Update-Systeme am Beispiel JDownloader

Projektarbeit

im Studiengang
Mechatronik

vorgelegt von

Thomas Rechenmacher
Matr.-Nr.: 2105101

am Montag, 23. September 2013
an der Friedrich-Alexander-Universität Erlangen-Nürnberg

Erstprüfer:	**Prof. Dr. Dirk Riehle**
Zweitprüfer/in:	**Prof. Dr.-Ing. Habil. Kai Willner**

Kurzfassung

JDownloader ist eine Open Source Projekt, das von Thomas Rechenmacher, dem Autor dieser Arbeit, und weiteren Entwicklern gegründet wurde. Die Anwendung erleichtert den Download von Dateien aus dem Internet. Das Projekt hat von 2007 bis heute die Entwicklung von einer Idee über ein Hobbyprojekt bis hin zu einem kommerziellen Hauptprodukt der Firma AppWork GmbH gemacht. Auf diesem Weg war es stets von großer Bedeutung die Anwendung über das Internet jederzeit aktualisieren zu können, um sie einerseits aktuell zu halten, aber auch, um Nutzer und Entwickler an das Projekt zu binden. Gegenstand dieser Arbeit ist das Online-Update-System wie es in JDownloader seit 2009 eingesetzt wird. Die detaillierte Analyse dieses Systems, dessen Anforderungen und dessen Realisierung, zeigt deutliche Mängel in der Planung und Umsetzung desselben. Basierend auf dieser Erkenntnis werden die Anforderungen neu formuliert, angepasst und erweitert, um schließlich als Grundlage und Hilfestellung für die Entwicklung eines neuen Update-Systems zu dienen.

Schlagwörter: JDownloader, Aktualisierung, Open-Source, Update-System, Anforderungen, Realisierung, Internet

Abstract

JDownloader is an Open Source file download software founded by Thomas Rechenmacher, the author of this study, and further developers. It has been evolving from an early hobby project in 2007 to what it is today, a major commercial product of the company “Appwork GmbH”. On this way, it has always been very important to be able to update the application through the internet in order to maintain a working copy of the program on the one hand, and to stay in contact to the users on the other hand. This study focuses on the online update system that has been in use in JDownloader since 2009. The detailed analysis of the system, its requirements and implementation, reveals significant deficiencies in planning and realizing the system. This insight provides the opportunity to have a closer look at old requirements, adapt them and add new ones to finally get a new basis for the next major JDownloader 2 update system.

Keywords: JDownloader, Update, Online, Open-Source, Requirements, Implementation, Internet

Inhaltsverzeichnis

Kurzfassung **2**

Abstract **2**

Inhaltsverzeichnis **3**

Abbildungsverzeichnis **5**

Tabellenverzeichnis **6**

Abkürzungsverzeichnis **7**

Vorwort **8**

1 Überblick **9**

2 JDownloader **11**

3 Ziele **13**

4 Stand der Technik **14**

4.1 Definitionen 14

4.2 Komponenten eines Update-Systems 14

4.2.1 Verwaltungsanwendung 15

4.2.2 Update-Server 15

4.2.3 Aktualisierungsanwendung 15

4.3 Inkrementell- oder Komplett-Updates 16

4.4 Signaturen 16

4.5 Marktübersicht 17

5 Das JDownloader Update-System **18**

5.1 Anforderungen 18

5.1.1 „Sackgassen“ verhindern 18

5.1.2 Konsistenter Datenbestand 18

5.1.3 Update-Zweige 19

5.1.4 Kurze Reaktionszeiten 19

5.1.5 Verwaltung 20

5.1.6 Server Anforderungen 20

5.1.7 Skalierbarkeit 20

5.1.8 Inkrementelles Update-Verfahren 20

5.1.9 Plugins zur Laufzeit ersetzen 21

5.1.10 Konnektivität 21

5.1.11 Programmiersprache ... 21
5.1.12 Betriebssysteme: Linux, Mac OS und Windows ... 21
5.1.13 Dateianzahl und Speicherbedarf ... 22
5.2 Realisierung ... 22
5.2.1 Serverstruktur ... 23
5.2.2 Verwaltungsanwendung ... 26
5.2.3 Aktualisierungsanwendung ... 30

6 Evaluation ... 39
6.1.1 Erfüllung der Anforderungen ... 39
6.1.2 Neue und geänderte Anforderungen ... 45

7 Zusammenfassung und Ausblick ... 49

Eidesstattliche Versicherung ... 51
Auszug aus dem Strafgesetzbuch (StGB) ... 51

Quellenverzeichnis ... 52

Abbildungsverzeichnis

Abbildung 1: Anzahl der Suchanfragen über die Zeit, die bei Google zum Suchbegriff „JDownloader“ eingingen [3]. Ein deutlicher Abwärtstrend setzt 2010 ein. 11
Abbildung 2: Einordnung von Update-Systemen in der Prozesskette in der Softwareentwicklung 14
Abbildung 3: Prozess der Softwareverteilung im JDownloader Projekt 16
Abbildung 4: Grundstruktur des Update-Systems 23
Abbildung 5: Tägliche Lastspitzen auf update4.jdownloader.org (Grafik bereitgestellt durch die AppWork GmbH [1]) 25
Abbildung 6: Lastverteilung auf mehrere Daten- und Kontroll-Server 26
Abbildung 7: Prozessübersicht: Eine neue Version verteilen. [5] 29
Abbildung 8: Aktualisierungsprozess (Teil 1) aus *JDownloader.jar* gestartet 33
Abbildung 9: Aktualisierungsprozess (Teil 2) aus *JDownloader.jar* gestartet 34
Abbildung 10: Lastenverteilung der Kontroll-Server 35
Abbildung 11: Lastverteilung der Daten-Server 36

Tabellenverzeichnis

Tabelle 1: Update-Systeme bekannter Anwendungen 17
Tabelle 2: Benennung der Listen auf den Kontroll-Servern 28
Tabelle 3: Entscheidungstabelle Betriebssystem Filter 37
Tabelle 4: Übersicht über die Anforderungserfüllung 39
Tabelle 5: Zeit für den Aufbau einer HTTP-Download Anfrage an update1.jdownloader.org 44

Abkürzungsverzeichnis

FTP[1]	File Transfer Protocol
HTTP[2]	Hypertext Transfer Protocol
SVN[3]	Subversion
MD5[4]	Message-Digest Algorithm 5
JAR	Java Archive
JVM	Java Virtual Machine

[1] „The objectives of FTP are […] to transfer data reliably and efficiently." [11]

[2] „The Hypertext Transfer Protocol (HTTP) is an application-level protocol for distributed, collaborative, hypermedia information." [12]

[3] „Subversion is a free/open source version control system […].That is, Subversion manages files and directories, and the changes made to them, over time." [23]

[4] „The algorithm takes as input a message of arbitrary length and produces as output a 128-bit "finger-print" or "message digest" of the input." [15]

Vorwort

Die Grundlage für diese Arbeit ist das, 2007 gegründete, Open-Source Projekt JDownloader. Als einer der Gründer habe ich das Projekt von Anfang an verfolgt, voran getrieben und weiter entwickelt. Wie das JDownloader Projekt, sind auch die Entwickler und ich, im Laufe der Jahre, an den immer neuen Herausforderungen gewachsen.

Wie viele andere Hobbyprojekte, ist auch JDownloader aus dem Wunsch heraus entstanden alles besser und offener zu gestalten. Mangels Erfahrung und bedingt durch eine ordentliche Portion Übermotivation wurde die Planungsphase damals einfach übersprungen. Wir haben einfach „losgelegt“. Aus Fehlern lernt man bekanntlich – und wir haben viel gelernt.

Das Resultat dieses Blitzstarts und der fehlenden Planungsschritte sind viele Iterationen. Immer wieder wurden Module komplett neu entworfen um Anforderungen zu erfüllen die wir erst erkannten nachdem die alten Module veröffentlicht wurden. Davon war auch das Update-System betroffen. Über zwei Jahre wurde an dem System „gebastelt“, das noch heute im Einsatz ist.

2009 ist aus dem JDownloader Projekt die Firma AppWork GmbH [1], an der ich 55% der Anteile halte, entstanden. Finanzielle Aspekte und Haftungsrisiken hatten diesen Schritt nötig gemacht. Seit Firmengründung bin ich für die AppWork GmbH als beherrschender Geschäftsführer und Entwickler tätig.

Leider gibt es kaum eine Dokumentation oder Protokolle zum Entstehungsprozess des ersten Update-Systems. Die besten Quellen um diesen nachvollziehen zu können sind einerseits die eigenen Erfahrungen und andererseits das Subversion Repository [2]. Dieses SVN-Quellcode-Repository enthält alle Änderungen und Commit-Kommentare[5] seit 2007, und erlaubt damit einen detaillierten Rückblick.

[5] Beim Speichern von Quellcodeänderungen auf dem Repository, können von den Entwicklern Kommentare zu den Änderungen abgegeben werden. Dadurch gibt es zu vielen Änderungen entsprechende Kommentare.

1 Überblick

Online-Update-Systeme sind ein sehr komfortables Werkzeug um Software über das Internet zu aktualisieren. Fehler oder Sicherheitslücken lassen sich so einfach beheben und neue Funktionen können verteilt werden. Die Software stellt dazu über das Internet eine Verbindung zu einem Webdienst her und kann von diesem die neuste Version von sich selbst laden und installieren. Der bekannte Downloadmanager „JDownloader" führte eine solche Update-Funktionalität schon sehr früh (September 2007) ein. Regelmäßige Aktualisierungen, auch von Alpha und Beta Versionen, waren von Anfang an ein fester Bestandteil des agilen Entwicklungsstils der JDownloader Community. Bei näherer Betrachtung zeigt sich allerdings, dass die zu Grunde liegende Update-Logik häufig geändert wurde.

Im September 2007 wurde ein dateibasierendes Update-System eingeführt. Über eine Liste von Prüfsummen wurde Datei für Datei verglichen und bei Änderungen die neue Version von einem Webdienst geladen. Dieses System ist zum Zeitpunkt der Veröffentlichung dieser Arbeit noch immer im Einsatz.

Im August 2012 wurde schließlich beschlossen, den dateibasierenden Vergleich aufzugeben, um einen neuen Ansatz zu wählen. Ein Webdienst sollte für jede Anfrage ein Paket liefern das alle Änderungen enthält. Im Januar 2013 wurde das neue Update-System in der JDownloader 2 Beta Version veröffentlicht.

Dieser Arbeit liegt das erste Update-System zu Grunde. Die Analyse des Systems verfolgt folgende Kernziele:

- Analyse der Anforderungen von JDownloader an ein Update-System
- Bereitstellung einer technischen Dokumentation zur Realisierung
- Rückblickende Evaluation des Systems

In Anbetracht dieser Ziele, die in Kapitel 3 näher erläutert werden, wird in Kapitel 2 zunächst das JDownloader Projekt näher beleuchtet. Eine Definition des Begriffs „Update-System" und ein kurzer Überblick marktüblicher Update-Systeme folgt im Kapitel 4.5 anhand einiger bekannter Beispiele.

Der Hauptteil der Arbeit beschäftigt sich, in Kapitel 5, mit der Beschreibung der Anforderungen, die JDownloader an ein Update-System stellt, sowie einer Dokumentation der technischen Umsetzung aller Kernkomponenten eines solchen Systems: Die Software für Server, Endnutzerrechner und die Verwaltung des Systems.

Schlussendlich werden Realisierung und Anforderungen rückblickend gegenübergestellt, um Fehler und Schwachstellen in eben diesen aufzudecken. Ein zusammenfassendes Fazit und ein Ausblick auf das, sich seit 2013 im Betatest befindende Nachfolgesystem, schließt die Arbeit in Kapitel 7 ab.

2 JDownloader

Das JDownloader Projekt wurde im Juli 2007 als Open Source Projekt unter der GPLv3 Lizenz gegründet. Die Entwickler setzten sich das Ziel, einen plattformunabhängigen Downloadmanager zu erstellen, der über sogenannte „Plugins", an komplizierte Downloadvorgänge, wie z.B. den Download von Sharehoster Seiten wie Rapidshare.com, angepasst werden konnte. Plugins automatisieren einen Downloadvorgang, indem sie dieselben HTTP-Anfragen tätigen, die ein Mensch beim manuellen Download im Browser auch tätigen würde. Die Eingaben und Mausklicks eines Endnutzers im Browser werden simuliert. Für jede unterstützte Webseite gibt es dazu ein eigenes Plugin. Änderungen an einer Webseite machen deswegen oft eine Änderung am Plugin nötig.

Aus diesem Grund waren von Anfang an sehr kurze Entwicklungszyklen notwendig. Es galt die Zeit zwischen Fehlermeldung und Veröffentlichung des aktualisierten Plugins, aber auch die Zeit bis eine solche Aktualisierung beim Endnutzer installiert werden konnte, kurz zu halten. Besonders in den ersten Monaten war die Plugin Schnittstelle nicht stabil. Das führte neben vielen Plugin-Aktualisierungen auch zu vielen Aktualisierungen der Kernkomponenten. Der Wunsch nach einer Möglichkeit Update-Pakete schnell und einfach zu veröffentlichen und zum Endnutzer zu bringen war geboren. Während anfangs noch Java Webstart eingesetzt wurde um den Endnutzern immer Zugriff auf die neuste Version zu bieten, wurde schon nach wenigen Monaten am ersten Online-Update-System gearbeitet.

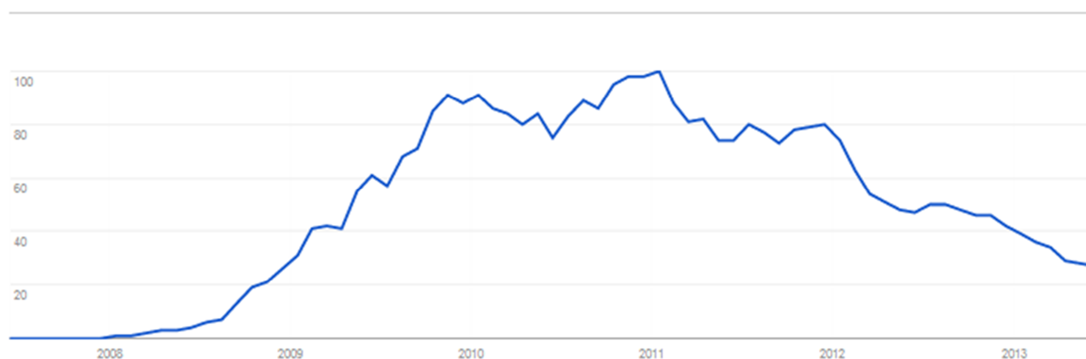

Abbildung 1: Anzahl der Suchanfragen über die Zeit, die bei Google zum Suchbegriff „JDownloader" eingingen [3]. Ein deutlicher Abwärtstrend setzt 2010 ein.

Im April 2009 ging aus dem JDownloader Projekt die Firma AppWork UG (haftungsbeschränkt) hervor und führte in weiten Teilen die Projektleitung fort. Ende 2009 wurde die Version 0.9581 als „Stable" Version veröffentlicht. Zu diesem Zeitpunkt gab es schon ca. 560 Plugins. Gleichzeitig wandte man sich von den kurzen Release-Zyklen

ab. Seit 2010 verzichtet man auf Aktualisierungen der Kernkomponenten und Plugins werden nur noch im Intervall von 1-2 Wochen veröffentlicht.

Das JDownloader Team arbeitet inzwischen an JDownloader 2. Um wieder zu den bewährten kurzen Release-Zyklen zurückkehren zu können, wurde 2012 ein neues Update-System eingeführt. Der öffentliche Betatest dieses Update-Systems läuft seit Januar 2013. Änderungen am Quellcode können damit wieder innerhalb weniger Minuten veröffentlicht und verteilt werden. Seit Veröffentlichung des neuen Systems gibt es im Durchschnitt zwischen 5 und 6 Updates täglich. Dieses Ziel konnte also erreicht werden.

3 Ziele

Ziel dieser Arbeit ist es, das erste Update-System, das im JDownloader Projekt zum Einsatz kam, zu analysieren, zu dokumentieren und schließlich zu evaluieren. Dazu soll nicht nur eine technische Dokumentation entstehen, sondern diese auch genutzt werden, um die Leistung des Systems in Hinblick auf dessen Anforderungen zu bewerten. Probleme in der Anforderungsanalyse selbst, aber auch in deren Realisierung, sollen so aufgedeckt werden.

Im Weiteren wird diese Arbeit aber auch hilfreich sein Fehler und Schwachstellen im System, der dahinterliegenden Logik und der Realisierung zu finden. Für ein Freeware Projekt wie JDownloader ist die Masse an Endnutzern sehr wichtig. Fehler im Update-System können nicht nur den Endnutzer von weiteren Aktualisierungen „abschneiden", sondern auch gefährliche Sicherheitslücken darstellen indem Dritten ermöglicht wird, Schadcode auf die Rechner der Endnutzer zu schleusen.

Schlussendlich soll diese Arbeit als Grundlage zu einem detaillierten Vergleich zwischen altem und neuem System dienen. Diese Analyse wird im Rahmen einer weiteren Arbeit erfolgen und sich intensiv mit den neuen Möglichkeiten, die sich aus dem Update-System 2 für Endnutzer und Entwickler ergeben, beschäftigen.

4 Stand der Technik

4.1 Definitionen

Die Hauptaufgabe eines Online-Update-Systems ist die Verteilung von neuen oder modifizierten Softwaremodulen. Damit setzt ein Update-System in der Prozesskette der Softwareentwicklung dort an, wo die Aufgaben von Systemen zur „Kontinuierlichen Auslieferung“, wie CruiseControl[6][7] oder Jenkins[8] enden.

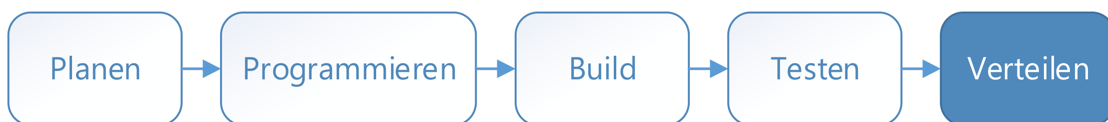

Abbildung 2: Einordnung von Update-Systemen in der Prozesskette in der Softwareentwicklung

Ein Update-System kann dabei nicht nur die Aufgabe übernehmen, eine bestehende Installation zu aktualisieren, sondern kann in der Rolle einer Installationsanwendung auch eine komplette Installation auf dem Zielrechner übernehmen. Die dazu benötigten Daten werden, wie bei einer Aktualisierung, über das Internet geladen.

4.2 Komponenten eines Update-Systems

Im Beispiel JDownloader bestehen alle Update-Systeme aus mehreren Komponenten (Abbildung 3, Seite 16). Eine Verwaltungsanwendung lädt eine neue Version auf einen oder mehrere Update-Server, von denen die Aktualisierungsanwendung auf den Rechnern der Endnutzer schließlich die Update-Pakete beziehen kann. Ein Update-Paket ist als eine Sammlung von Dateien und Daten zu verstehen. Der Container dieser Sammlung spielt dabei keine Rolle. Update-Pakete können beispielsweise einzelne Dateien, aber auch komprimierte Archive sein. Das Installieren eines Update-Pakets aktualisiert eine Anwendung von einem veralteten auf einen neueren Stand.

---

[6] Im Falle von JDownloader ist es ein Ziel der kontinuierlichen Auslieferung, jederzeit eine neue Version automatisiert verteilen zu können und dies auch möglichst häufig zu tun.

[7] „CruiseControl is both a continuous integration tool and an extensible framework for creating a custom continuous build process“ [9]

[8] „An extendable open source continuous integration server” [7]

4.2.1 Verwaltungsanwendung

Über eine Verwaltungsanwendung werden die Datenbestände auf den Update-Servern gepflegt und verwaltet. Steht eine neue Version bereit, wird sie über die Verwaltungsanwendung an die Server verteilt. Alle Prozessschritte, die dabei nicht auf einem Update-Server laufen dürfen, müssen auf dem Rechner des Herstellers ausgeführt werden, und fallen damit in das Aufgabengebiet der Verwaltungsanwendung. Dazu zählen z.B. sicherheitsrelevante Schritte wie das Signieren von Paketen und Dateien.

4.2.2 Update-Server

Der Update-Server ist der Kontrollpunkt im System. Er stellt die Schnittstelle zur Kommunikation für die Aktualisierungsanwendung und die Verwaltungsanwendung bereit. Die Kernaufgabe des Update-Servers besteht darin, Anfragen der Aktualisierungsanwendungen zu beantworten. Die Serverstrukturen sind häufig redundant ausgelegt, um Ausfälle zu verhindern. Der Begriff „Server" ist dabei sehr abstrakt zu sehen. Als Update-Server kann ein dedizierter Rechner, ein virtueller Server, ein Cloud-Dienst wie Amazon CloudFront[9] oder andere Hosting Dienste zum Einsatz kommen. Entscheidend sind dabei vor allem

- Die Fixkosten
- Die variablen Kosten z.B. für genutztes Transfervolumen
- Die zur Verfügung stehende Bandbreite und Rechenleistung
- Die Ausfallsicherheit
- Die Erreichbarkeit rund um die Welt

4.2.3 Aktualisierungsanwendung

Die Aktualisierungsanwendung läuft auf den Rechnern der Endnutzer und kontaktiert den Webdienst regelmäßig. Meldet dieser, dass eine Aktualisierung geladen werden kann, wird das entsprechende Paket heruntergeladen und installiert. Neben dem Laden und Installieren neuer Update-Pakete ist es eine der wichtigsten Aufgaben der Aktualisierungsanwendung sich selbst fehlerfrei aktualisieren zu können. Stellt der Update-Server Signaturen für die geladenen Pakete zur Verfügung, sollten diese vor der Installation verifiziert werden.

---

[9] „Amazon CloudFront ist ein Web-Service für die Bereitstellung von Inhalten." [4]

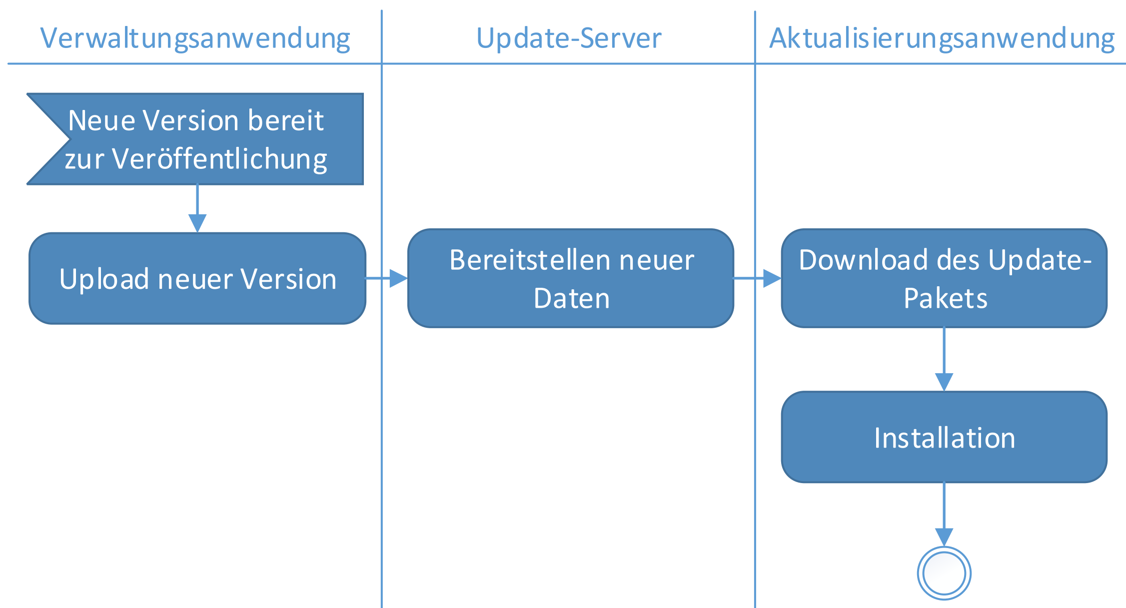

Abbildung 3: Prozess der Softwareverteilung im JDownloader Projekt

4.3 Inkrementell- oder Komplett-Updates

Grundsätzlich muss man zwischen einem Komplett-Update und einem inkrementellen Update-Verfahren unterscheiden. Bei einem Komplett-Update wird die komplette Installationsdatei geladen. Diese Installationsdatei enthält bei Komplett-Updates alle Ressourcen der Anwendung. Die alte Version wird also komplett ersetzt oder überschrieben. Dabei werden selbst Dateien ausgetauscht, die sich nicht geändert haben. Dieses Verfahren ist für JDownloader ungeeignet, da sich in den meisten Fällen nur ein kleiner Teil der Anwendung tatsächlich geändert hat. Komplett-Updates würden für unverhältnismäßig lange Download- und Installationszeiten sowie für zu viel Transfervolumen sorgen. Für einen kompletten Austausch der Daten müsste außerdem die Hauptanwendung beendet werden. Der Endnutzer wäre folglich gezwungen seine Arbeit mit der Anwendung zu unterbrechen.

Beim inkrementellen Verfahren werden nur geänderte Teile erneuert. Unveränderte Teile der Anwendung werden nicht geladen oder installiert. Daraus ergeben sich kurze Downloadzeiten und geringere Kosten für den Datenverkehr. Inkrementelle Aktualisierungen können teils sogar zur Laufzeit installiert werden. Der Endnutzer könnte dann die Anwendung während des Downloads und der Installation weiter ungestört nutzen. Diese Vorteile werden allerdings mit deutlich höherem Rechen– und Speicheraufwand beim Erstellen und Speichern der Update-Pakete erkauft.

4.4 Signaturen

Bei einer Aktualisierung über das Internet wird neuer Programmcode über das Internet von einem Update-Server geladen und installiert. Erlangt ein Angreifer die Kontrolle

über den Update-Server oder kann den Datenfluss zwischen Endnutzer und Update-Server manipulieren (Man-In-The-Middle Attacke), so kann der Angreifer über das Update-System Schadcode auf alle Endnutzer verteilen. Um das zu verhindern, sollten alle Update-Pakete signiert werden.

Der Hersteller erstellt dazu über die Verwaltungsanwendung (4.2.1) eine Signatur für jedes Update-Paket. Beim Signieren wird dazu eine digitale Unterschrift zu einer Datei erstellt. Die Unterschrift kann anschließend auf Echtheit und Gültigkeit überprüft werden. Die Aktualisierungsanwendung installiert ein Paket nur nach erfolgreicher Signaturprüfung. Durch die Überprüfung kann sichergestellt werden, dass das installierte Paket auch wirklich vom Hersteller der Software kommt, und nicht von Dritten modifiziert wurde.

4.5 Marktübersicht

In den letzten Jahren ist zu beobachten, dass neue Betriebssysteme zunehmend mit integrierter Paket- und Softwareverwaltung veröffentlicht werden. Da für JDownloader eine plattformunabhängige Lösung nötig ist, kommen diese allerdings nicht in Frage. Eine Recherche zeigt, dass es kaum marktfertige Systeme gibt, die einerseits automatisierte inkrementelle Aktualisierungen ermöglichen und andererseits einfach plattformübergreifend integrierbar sind. In der Praxis verzichten viele Anwendungen komplett auf Aktualisierungen über das Internet, oder nutzen einfache Komplett-Update-Systeme. Erschreckenderweise fehlt bei einigen verbreiteten Anwendungen jegliche Signatur. Ein erfolgreicher Angreifer auf deren Serverstruktur hätte Zugriff auf Millionen von Rechnern. Tabelle 1 zeigt am Beispiel einiger bekannter Anwendungen grundlegende Funktionen von deren Update-Systemen.

	Online-Update	Paketierung	Signatur	Verschlüsselung
NotePad++ 6.0	Ja	Komplett-Update	Nein	Nein
Paint.net 3.5.10	Ja	Komplett-Update	Nein	Nein
VLC Player	Ja	Komplett-Update	Ja	Nein
Wireshark 1.8.3	Nein	—	—	—
Winrar	Nein	—	—	—

Tabelle 1: Update-Systeme bekannter Anwendungen

Da es damals wie heute keine frei zugänglichen Update-Systeme auf dem Markt gibt, die den Anforderungen von JDownloader genügen, wurde ein eigenes entwickelt.

5 Das JDownloader Update-System

Das erste Update-System wurde sehr bald nach Projektstart geplant und ab September 2007 umgesetzt. Es wurde bis zur Veröffentlichung von JDownloader 0.9581, Ende 2009, mehrmals erweitert, um den immer größer werdenden Anforderungen zu genügen. Im Folgenden wird das Update-System in der Revision 9851 (Stand 25.11.2009) analysiert. Der analysierte Quellcode kann vom JDownloader Subversion Repository [2] bezogen werden.

5.1 Anforderungen

JDownloader wurde als Hobbyprojekt geboren und wird seit dem stetig weiter entwickelt. Mit steigendem Erfolg und steigenden Nutzerzahlen änderten sich die Anforderungen sehr rasch. Die wichtigsten Anforderungen an das System werden im Folgenden beschrieben.

5.1.1 „Sackgassen“ verhindern

Die Endnutzerzahlen sind das wichtigste „Kapital“ des JDownloader Projekts. Bei jedem Open-Source Projekt besteht eine große Chance, dass Endnutzer auf ein abgespaltenes Projekt (Fork) wechseln. Die Bindung von Endnutzern an das eigene Projekt war folglich eines der wichtigsten Ziele des Update-Systems. Entwickler sollten durch die große Nutzerzahl des Hauptprojekts animiert werden daran mitzuwirken anstatt ein abgeleitetes Projekt zu starten. Es dürfen also keine Endnutzer „verloren“ gehen.

Die wichtigste Anforderung an das System war es „Sackgassen“ zu vermeiden. Eine Sackgasse ist, im JDownloader Kontext, eine Situation in der das Update-System sich selbst aussperrt. So eine Situation entsteht beispielsweise, wenn das Update-System sich selbst aktualisiert, und dabei die eigenen Routinen beschädigt. Solche Situationen mussten vermieden oder verhindert werden. Das System sollte jederzeit in der Lage sein, zumindest sich selbst, möglichst ohne initiale Endnutzerinteraktion, zu aktualisieren.

5.1.2 Konsistenter Datenbestand

Unter einem inkonsistenten Zustand versteht man einen Dateistand der aus einer Mischung von zueinander inkompatiblen Dateien besteht. Werden manche Programmteile aktualisiert, während andere auf einem veralteten Stand bleiben, kann es zu einer solchen Situation kommen. Die Aktualisierungsanwendung muss sicherstellen, dass solche Fehler nicht entstehen.

Um das zu gewährleisten müssen, unter anderem, folgende Operationen möglich sein:

- Durch den Benutzer modifizierte Dateien erkennen und ersetzten.
- Veraltete Dateien erkennen und ersetzen.
- Nicht mehr benötigte Dateien löschen.
- Neue Dateien installieren.

5.1.3 Update-Zweige

Das System musste das Zuteilen von Paketen an bestimmte Gruppen ermöglichen. Die Anzahl der Gruppen durfte dabei nicht beschränkt sein. Um den Endnutzer den Wechsel zwischen verschiedenen Zweigen zu ermöglichen, war es nötig, dass die Aktualisierungsanwendung diese Funktion bereitstellt. Zur Verdeutlichung folgen beispielhaft zwei typische Anwendungsfälle:

Anwendungsfall 1:

Ein neues Update-Paket wird beispielsweise zuerst für eine „ALPHA“ Gruppe freigegeben. Nach erfolgreichen Tests in der „ALPHA“ Gruppe, könnte das Paket an die Gruppe „ALLE“ verteilt werden.

Anwendungsfall 2:

Ein weiterer Anwendungsfall ist das Anbieten eines „NIGHTLY“-Zweigs. In diesem Zweig können täglich automatisierte Builds als Momentaufnahme des aktuellen Entwicklungsstandes angeboten werden.

5.1.4 Kurze Reaktionszeiten

In Kapitel 2 wurde bereits erläutert, dass Plugins eine wichtige Komponente von JDownloader sind. Sie erweitern die Anwendung und erleichtern den Download von vielen Quellen. Das „RapidShare.com“-Plugin ermöglicht es JDownloader beispielsweise, von diesem Online-Speicherplatzanbieter automatisiert und ohne Endnutzerinteraktion zu laden. Änderungen an der unterstützten Webseite machen meist eine Anpassung des betroffenen Plugins erforderlich. Darüber hinaus versuchen viele Webseitenbetreiber JDownloader durch gezielte Änderungen an der Webseite zu sperren. Aus diesem Grund müssen Plugins häufiger als andere Programmteile aktualisiert werden. Es ergibt sich daraus folgende Anforderung:

Das Update-System muss in der Lage sein, jederzeit, in einem Zeitrahmen von höchstens 30 Minuten, ein neues Update zu verteilen.

Abgesehen von Plugin-Aktualisierungen sind kurze Reaktionszeiten auch nötig, um Fehler so schnell wie möglich ausbessern zu können. Im agilen Entwicklungsstil des

JDownloader Teams wurden die Endnutzer als Tester herangezogen. Im Falle von negativem Nutzer-Feedback oder Fehlermeldungen war ein Weg erforderlich, um die erneuerten Programmteile schnell zu verteilen.

5.1.5 Verwaltung

Die benötigte Zeit um eine neue Version zu veröffentlichen sollte so kurz wie möglich sein. Außerdem war eine weitgehende Automatisierung des ganzen Prozesses gewünscht, um menschliche Fehler beim Veröffentlichen zu minimieren. Die Verwaltungsanwendung würde nur von den Kernentwicklern des JDownloader Entwicklerteams genutzt werden und musste so keine besonderen Anforderungen an die Benutzerfreundlichkeit erfüllen. Eine grafische Benutzeroberfläche war folglich explizit nicht gefordert.

5.1.6 Server Anforderungen

JDownloader war 2009 ein nicht kommerzielles Open-Source Projekt. Als solches war das Team auf gespendete Server angewiesen. Um möglichst unkompliziert neue Server hinzufügen zu können, sollte es ausreichen, wenn auf den Servern Standardsoftware (FTP-Dienst, HTTP-Dienst) lief. Das Update-System musste also ohne eigene serverseitige Software auskommen. Die Anbindung anderer Hosting-Dienstleister wie „Google Code[10]“ oder „Amazon CloudFront“[11] musste ebenfalls möglich sein, um flexibel in der Anbindung weiterer und alternativer Update-Server zu sein.

5.1.7 Skalierbarkeit

Bereits 2009 nutzten mehrere Millionen Nutzer JDownloader. Eine Trendwende war noch nicht in Sicht (vgl. Abbildung 1, Seite 11). Das gesamte Update-System musste also hoch skalierbar ausgelegt werden, um auch zukünftigen Nutzerzahlen gerecht zu werden. Neben den Nutzerzahlen war auch die weitere Entwicklung des JDownloader Projekts unsicher. Resultierend aus einer möglichen drastischen Änderung der reinen Datenmenge einer Installation, musste mit einer deutlichen Änderung der benötigten Bandbreite und des benötigten Transfervolumens gerechnet werden.

5.1.8 Inkrementelles Update-Verfahren

Um den agilen Entwicklungsstil zu unterstützen, waren viele kleine Update-Pakete statt wenigen großen geplant. Wie bereits in Kapitel 4.3 erläutert wurde, eignet sich dazu ein inkrementelles Verfahren eher als Komplett-Updates. Darüber hinaus sollte Transfervo-

---

[10] „Project Hosting on Google Code provides a free collaborative development environment for open source projects.” [5]

[11] „Amazon CloudFront ist ein Web-Service für die Bereitstellung von Inhalten“ [4]

lumen gespart und der Aktualisierungsprozess für den Endnutzer beschleunigt werden. Inkrementelle Update-Pakete sollten nur aus Dateien bestehen die sich geändert haben.

5.1.9 Plugins zur Laufzeit ersetzen

Da Plugins in der Regel viel häufiger aktualisiert werden müssen als andere Programmteile, sollte es möglich sein Plugins getrennt vom Programmkern zu aktualisieren und zur Laufzeit austauschen zu können. Der Benutzer durfte nicht durch Plugin-Aktualisierungen zum Neustart von JDownloader gezwungen werden. Diese Anforderung erforderte das Zusammenspiel aus Aktualisierungsanwendung und dem Plugin-System in JDownloader. Zum Zeitpunkt der Planungsphase war diese Funktionalität noch nicht im Plugin-System verankert.

5.1.10 Konnektivität

Firewalls und Anwendungen zur Bekämpfung von Viren schränken auf vielen Computersystemen den Zugang ins Internet stark ein. Darüber hinaus können viele Endnutzer nur über einen Proxy-Dienst auf das Internet zugreifen. Um Problemen mit Firewalls aus dem Weg zu gehen und gängige HTTP-Proxy-Verbindungen nutzen zu können, war HTTP als Kommunikationsprotokoll zwischen Update-Server und Aktualisierungsanwendung gefordert. Firewalls lassen ausgehende Verbindungen auf HTTP-Port 80 meist zu. Die meisten Proxy-Verbindungen sind für HTTP-Verbindungen über Port 80 ebenfalls korrekt konfiguriert.

5.1.11 Programmiersprache

JDownloader 0.9851 ist in Java geschrieben, und benötigt eine Java Virtual Environment 1.5 oder neuer. Damit läuft JDownloader auf allen Geräten, für die Java ab Version 1.5 zur Verfügung steht. Folglich sollte die Aktualisierungsanwendung im selben Maße plattformübergreifend umgesetzt werden. Als Programmiersprache wurde Java gefordert.

5.1.12 Betriebssysteme: Linux, Mac OS und Windows

Im Kern ist JDownloader eine Java Anwendung, und damit, theoretisch, für alle Systeme gleich. Für einige Funktionen werden jedoch native Anwendungen und Bibliotheken verwendet. Das Update-System musste sicherstellen, dass an die verschiedenen Betriebssysteme gezielt unterschiedliche Ressourcen verteilt werden konnten.

Beispiel:

JDownloader kann Rar-Archive nach dem Download entpacken. Dazu wird die native Anwendung „unrar“ genutzt. Während unter Windows unrar.exe genutzt wird, benötigen Linux und Mac OS jeweils ihre eigenen Unrar-Binärdateien.

5.1.13 Dateianzahl und Speicherbedarf

JDownloader 0.9851 bestand aus ca. 840 Dateien, 110 Ordnern und 560 Plugin-Dateien. Der Speicherbedarf lag bei ca. 50 Megabyte. Eine konkrete Obergrenze für die Anzahl der Dateien wurde nicht vereinbart. Es galt ein System zu schaffen, das die bestehende Menge an Dateien und Daten problemlos verarbeiten kann.

5.2 Realisierung

Die Anforderungen wurden über einen Entwicklungszeitraum von zwei Jahren schrittweise implementiert. Die Entwicklung folgte dabei keinem klaren Weg. Über viele Iterationsschritte ist das System langsam aus einer Idee heraus gewachsen. Im Laufe dieses Entwicklungsprozesses wurden auch immer wieder neue Anforderungen gestellt, geändert und verworfen. Das System basierte schließlich auf einer Liste aller Dateipfade und deren Prüfsummen. Nachdem von einem Deployment-Rechner[12] diese Dateiliste erstellt wurde, wird diese zusammen mit der kompletten Ordner– und Dateistruktur auf die Update-Server geladen. Dazu wird bevorzugt das RSYNC[13] Protokoll, alternativ aber auch das „File Transfer Protocol(FTP)“ verwendet. Ein Update-Server kann dabei die Rolle eines Daten-Servers, die Rolle eines Kontroll-Servers oder auch beide Rollen übernehmen. Während ein Daten-Server zum Bereitstellen der reinen Nutzdaten dient, verwaltet ein Kontroll-Server die Datei- und Serverlisten. In der Praxis sind die Kontroll-Server unter

- *update0.jdownloader.org,*
- *update1.jdownloader.org und*
- *update2.jdownloader.org*

erreichbar, und übernehmen gleichzeitig auch die Rolle eines Daten-Servers. Die Adressen der Kontroll-Server sind fest in der Aktualisierungsanwendung hinterlegt.

---

[12] In der Praxis werden im JDownloader Team neue Versionen von einer virtuellen Maschine aus veröffentlicht. Auf dieses virtuelle System haben nur die Hauptentwickler der Firma AppWork GmbH Zugriff.

[13] „rsync is an open source utility that provides fast incremental file transfer.” [8] [2]

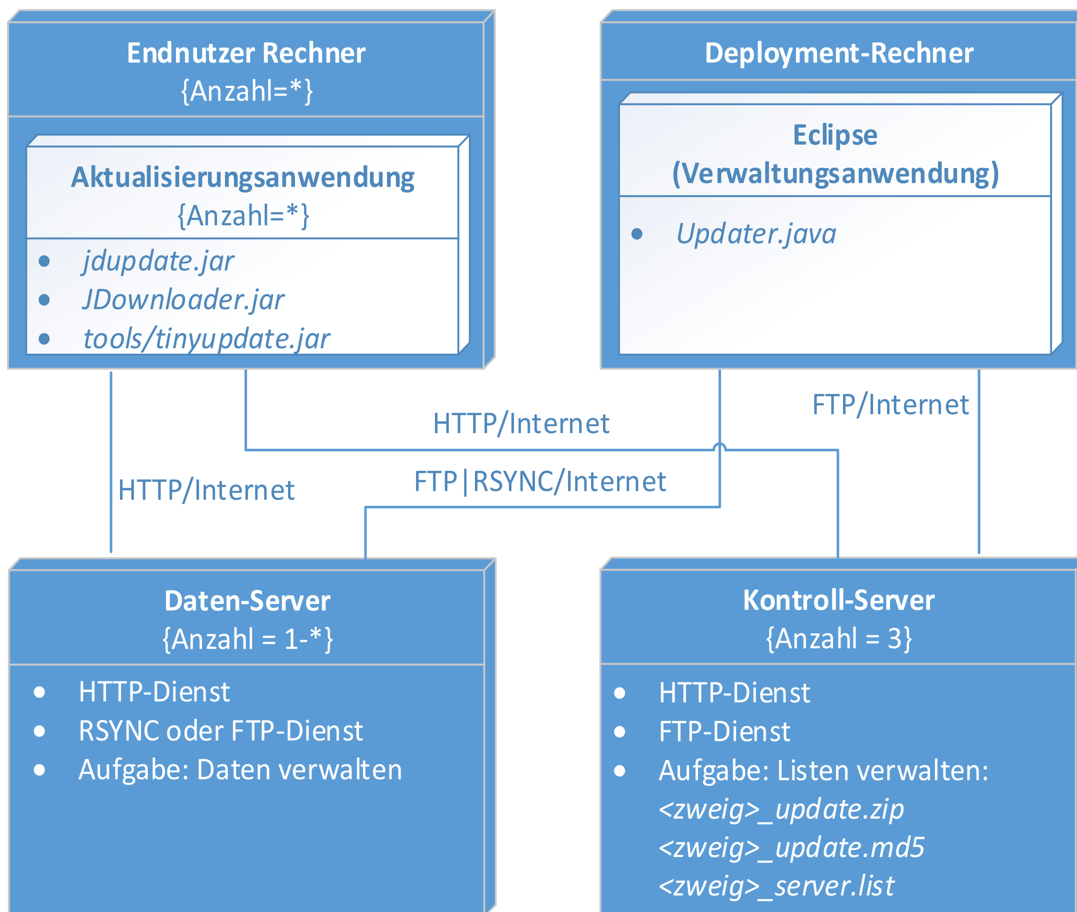

Abbildung 4: Grundstruktur des Update-Systems

Die Aktualisierungsanwendung auf dem Rechner der Endnutzer lädt die Dateiliste über das Internet von einem Kontroll-Server und gleicht den lokalen Datenbestand mit der Dateiliste ab. Werden modifizierte oder neue Dateien gefunden, so werden diese einzeln von den Daten-Servern geladen und installiert. Um veraltete Dateien zu finden und zu löschen, wird eine „*outdated.dat*"-Datei [4] gepflegt, die alle zu löschenden Pfade enthält.

5.2.1 Serverstruktur

Aufgrund der Anforderung mit Servern zu arbeiten, auf denen keine eigene Software installiert werden kann, musste das System mit serverseitiger Standardsoftware auskommen. Als Mindestanforderung muss ein HTTP-Dienst (Apache, Nginx, Lighttpd,…) zur Kommunikation mit den Aktualisierungsanwendungen und ein FTP- bzw. RSYNC-Dienst für den Datenaustausch mit der Verwaltungsanwendung installiert sein. Diese Voraussetzung erfüllen die meisten Hosting-Angebote im Netz.

Steht ein neues Update-Paket bereit, lädt jede Aktualisierungsanwendung die Dateiliste von einem Kontroll-Server, und anschließend jede geänderte Datei einzeln von einem Daten-Server. Über eine Prüfsummendatei der Dateiliste wird ein unnötiges Aktualisie-

ren der Liste verhindert. Das dadurch gesparte Transfervolumen wird durch eine zusätzliche Anfrage für den Download der Prüfsumme erkauft. Der HTTP-Dienst muss also pro JDownloader Installation zwei Anfragen für die Dateiliste (Prüfsumme + Liste) und eine Anfrage pro geänderter Datei beantworten.

Beispiel:

Nimmt man beispielsweise an, für die Aktualisierung auf eine neue Version müssten 500 Dateien(D) ersetzt werden, und am ersten Tag nach der Veröffentlichung dieser neuen Version würden 3 Millionen JDownloader Installationen(I) aktualisieren wollen, so müsste der Server, an diesem Tag, 1,5 Milliarden Anfragen beantworten.

$$I = 3000000 \frac{Aktualisierungen}{Tag}$$
$$D = 500\ Dateien$$
$$A = I \cdot 2 + I \cdot D\ (Anzahl\ der\ Anfragen)$$
$$A = I \cdot (2 + D)$$
$$A = 3000000 \cdot (500 + 1 + 1) \frac{Anfragen}{Tag}$$
$$A = 1506000000 \frac{Anfragen}{Tag} \approx 17431 \frac{Anfragen}{Sekunde}$$

Die durchschnittliche Dateigröße in JDownloader 0.9851 beträgt ca. 30 Kilobytes. Folglich wäre eine durchschnittliche Bandbreite von über 4 Gigabit/Sekunde nötig.

$$S_{avg} = 30000\ Byte$$
$$B = S_{avg} \cdot A \frac{Byte}{Sekunde}$$
$$B = 30000 \cdot 17431 \frac{Byte}{Sekunde}$$
$$B = 522930000 \frac{Byte}{Sekunde} = 4183440000 \frac{Bit}{Sekunde} \approx 4{,}2 \frac{Gigabit}{Sekunde}$$

Die tatsächlich benötigte Bandbreite ist aus folgenden Gründen tatsächlich deutlich höher:

- Durch die geographische Verteilung aller JDownloader Nutzer ergibt sich eine tägliche Lastspitze in den Abendstunden mitteleuropäischer Zeit. Erfahrungsgemäß ist die Last zu solchen Spitzenzeiten um mehr als 100% höher als die Durchschnittslast (Abbildung 5, Seite 25).

- Neben den Aktualisierungsprozessen prüfen alle JDownloader Installationen mindestens einmal beim Start ob sich die Dateiliste geändert hat. Dazu laden sie die MD5-Prüfsumme der Dateiliste vom Server und gleichen sie mit der gespeicherten Dateiliste ab. Dadurch kommt es zu zusätzlichen Anfragen von Installationen die bereits auf dem aktuellsten Stand sind.
- Der HTTP- und TCP/IP Overhead[14] wurde in der Rechnung nicht beachtet.

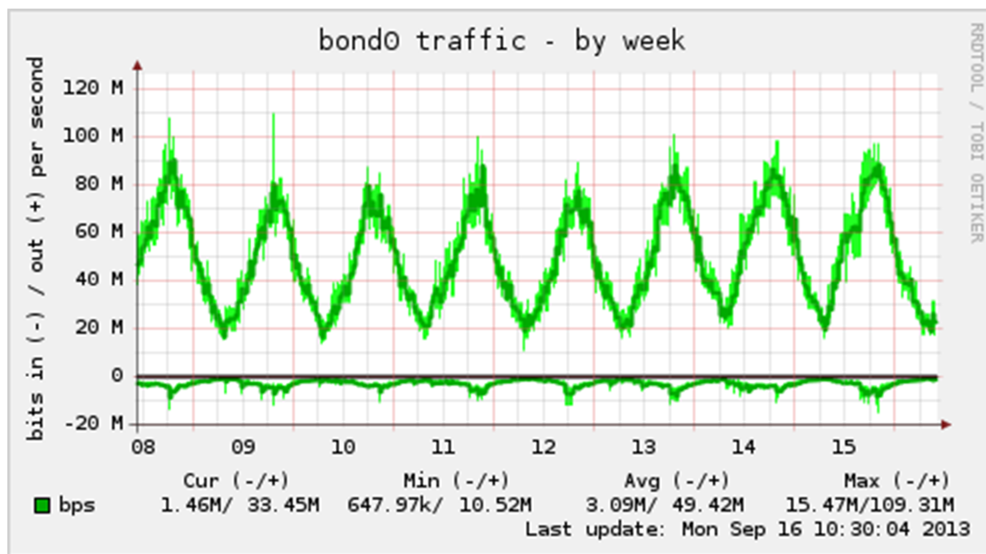

Abbildung 5: Tägliche Lastspitzen auf update4.jdownloader.org (Grafik bereitgestellt durch die AppWork GmbH [1])

Statt einer schnellen Anbindung hat man sich für mehrere getrennte langsame angebundene Server entschieden. Ausschlaggebend war dabei:

- Die hohe Ausfallsicherheit durch die redundante Server-Struktur;
- Die höhere Flexibilität – Die Bandbreite kann je nach Bedarf durch zusätzliche Server erhöht werden;
- Die Kosten. Schnellere dedizierte Anbindungen sind besonders im Ausland sehr teuer;

Marktübliche Miet-Server-Angebote ohne teure Backbone Anbindung bewegen sich zwischen 100 Megabit/Sekunde und 2 Gigabit/Sekunde. Es bietet sich also an die Last auf mehrere Server zu verteilen. Dazu wird der Datenbestand von der Verwaltungsanwendung auf mehrere Daten-Server gespiegelt. Eine Liste mit den zur Verfügung stehenden Daten-Servern kann von einem der Kontroll-Server geladen werden. Neben den Adressen der Daten-Server enthält diese Liste zusätzlich einen Prioritätsfaktor über den

[14] Protokoll Overhead Daten die nicht zu den Nutzdaten zählen, sondern als Zusatzinformationen wie Steuerzeichen oder Header für die Kommunikation selbst benötigt werden. Für das Hypertext Transfer Protocol (HTTP) können z.B. die HTTP-Header in der Anfrage und der Antwort als Protokoll Overhead bezeichnet werden.

die Lastverteilung auf die verschiedenen Server gesteuert werden kann. Einen schwächeren Server kann man über diese Prioritätsfaktoren prozentual entlasten.

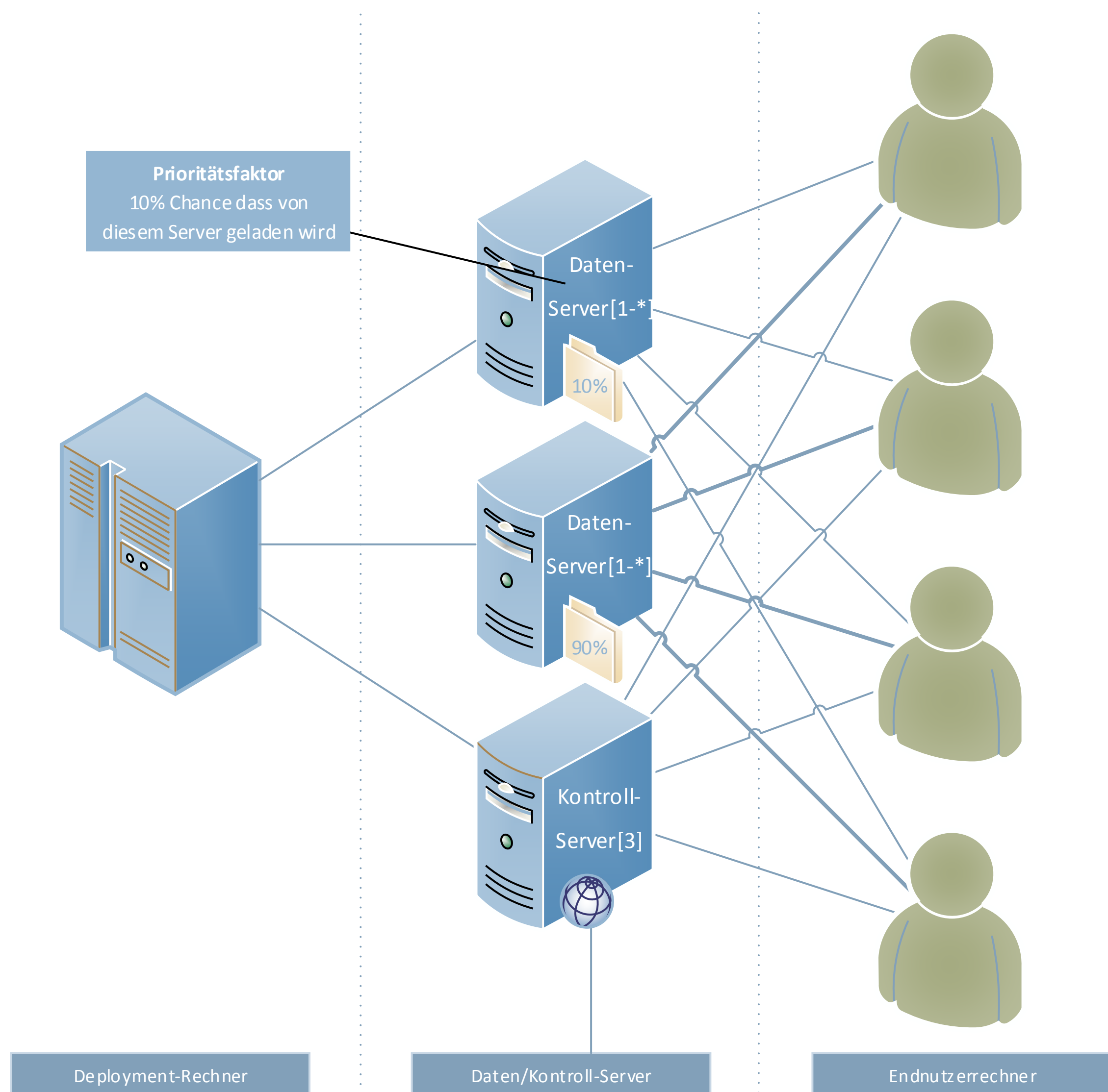

Abbildung 6: Lastverteilung auf mehrere Daten- und Kontroll-Server

5.2.2 Verwaltungsanwendung

Da die Update-Server keine eigene Programmlogik enthalten dürfen (Kapitel 5.1.6, Seite 20: Server Anforderungen), muss die Verwaltungsanwendung ein neues Update-Paket aufbereiten, paketieren und schlussendlich über FTP oder RSYNC auf die Daten- und Kontroll-Server laden.

Zur Verwaltung wurde keine eigenständige Anwendung[15] geschrieben, sondern eine Java Klasse (*Updater.java* [5]), die aus der Entwicklungsumgebung Eclipse gestartet

[15] Im JDownloader Entwicklerteam wird die Verwaltungskomponente in der Regel „Updateskript“ genannt.

wird. Statt einer grafischen Benutzeroberfläche werden Parameter direkt im Programmcode angepasst.

Soll eine neue Version veröffentlicht werden, müssen zuerst einige Einstellungen und Vorbereitungen vorgenommen werden:

1. Der gewünschte Update-Zweig (Kapitel 5.1.3, Seite 19: Update-Zweige)
2. Die Liste der Daten-Server inkl.
    - HTTP-Adressen unter denen später auf die Daten zugegriffen werden kann,
    - Upload Informationen und Zugangsdaten (FTP oder RSYNC) und
    - Prioritätsfaktor um die Auslastung der Server zu kontrollieren.
3. Der lokale Ordner aus dem das Skript die neuen Daten beziehen kann. Dieser Ordner wird im Folgenden als „Datenordner“ bezeichnet.
4. Dateien und Ordner die gelöscht werden sollen, müssen manuell in die Datei *<Datenordner>/outdated.dat* [4] eingetragen werden.

Mit diesen Informationen kann der Prozess gestartet werden. Bis zur Veröffentlichung der neuen Version werden folgende Schritte durchlaufen (vgl. Abbildung 7):

1. **Aktualisieren**:
   Zunächst wird die letzte Version des gewählten Update-Zweigs in einen temporären Ordner geladen.
2. **Kopieren**:
   Die neuen Dateien aus dem Datenordner werden ebenfalls in diesen temporären Ordner kopiert und ersetzen damit bereits vorhandene Dateien.
3. **Aufräumen**:
   Die Datei *<Datenordner>/outdated.dat* wird eingelesen und alle gefundenen Dateien und Ordner aus dem temporären Ordner gelöscht.
4. **Komprimieren**:
   Einige vordefinierte Ordner (z.B. die Übersetzungen) werden zu Zip-Archiven (**.extract* Dateien) komprimiert. Das spart später Bandbreite und reduziert die Anzahl der Dateien erheblich.
5. **Dateiliste laden**:
   Die Dateiliste mit den Prüfsummen der letzten Version wird erneut von einem der Kontroll-Server geladen.
6. **Filtern**:
   Die alte Dateiliste wird über die Prüfsummen mit allen Dateien im temporären Ordner verglichen. Alle modifizierten oder neuen Dateien werden in einer neuen Liste gesammelt. Nach diesem Schritt liegt eine neue Dateiliste mit Prüfsummen vor.
7. **Daten Upload**:

Alle Dateien im temporären Ordner werden nun der Reihe nach auf alle Daten-Server verteilt. Die Dateien werden dabei in einen Unterordner mit dem Namen des gewählten Update-Zweigs verschoben. Der vollständige Pfad der Datei *outdated.dat* wird so zu:

→*http://<Update-Server-HTTP-Adresse>/<Update-Zweig>/outdated.dat.*

Zur Übertragung der Dateien werden die Protokolle RSYNC oder FTP verwendet.

8. **Listen Upload**:
Das Update-System kennt drei Kontroll-Server (Kapitel 5.2.1, Seite 23: Serverstruktur). Auf diese Server wird die neue Dateiliste mit den Prüfsummen geladen. Diese Liste enthält für jede Datei den relativen Pfad sowie die zugehörige MD5-Prüfsumme. Außerdem wird von der Dateiliste ebenfalls die MD5-Prüfsumme errechnet und, als *<Update-Zweig>_update.md5* Datei, auf den Servern abgelegt. Schlussendlich wird noch die Liste aller Daten-Server auf jedem Kontroll-Server gespeichert. Für die Benennung der Dateien gilt folgendes Schema:

Beschreibung	Dateiname
Dateiliste (im Zip-Archiv)	*<Update-Zweig>_update.zip*
Prüfsumme der komprimierten Dateiliste	*<Update-Zweig>_update.md5*
Liste der Update-Server	*<Update-Zweig>_server.list*

Tabelle 2: Benennung der Listen auf den Kontroll-Servern

Nachdem der Prozess beendet wurde, ist jeder Kontroll- und Daten-Server auf demselben Datenstand und stellt die Daten für die Aktualisierungsanwendungen bereit. Folgende Dateien können von den verschiedenen Update-Servern abgerufen werden:

Kontroll-Server

- Dateiliste *./<Update-Zweig>_update.zip*
- Prüfsumme der Dateiliste *./<Update-Zweig>_update.md5*
- Liste aller Daten-Server *./<Update-Zweig>_server.list*

Daten-Server

- Alle Dateien und Ordner im Unterordner *./<Update-Zweig>/*

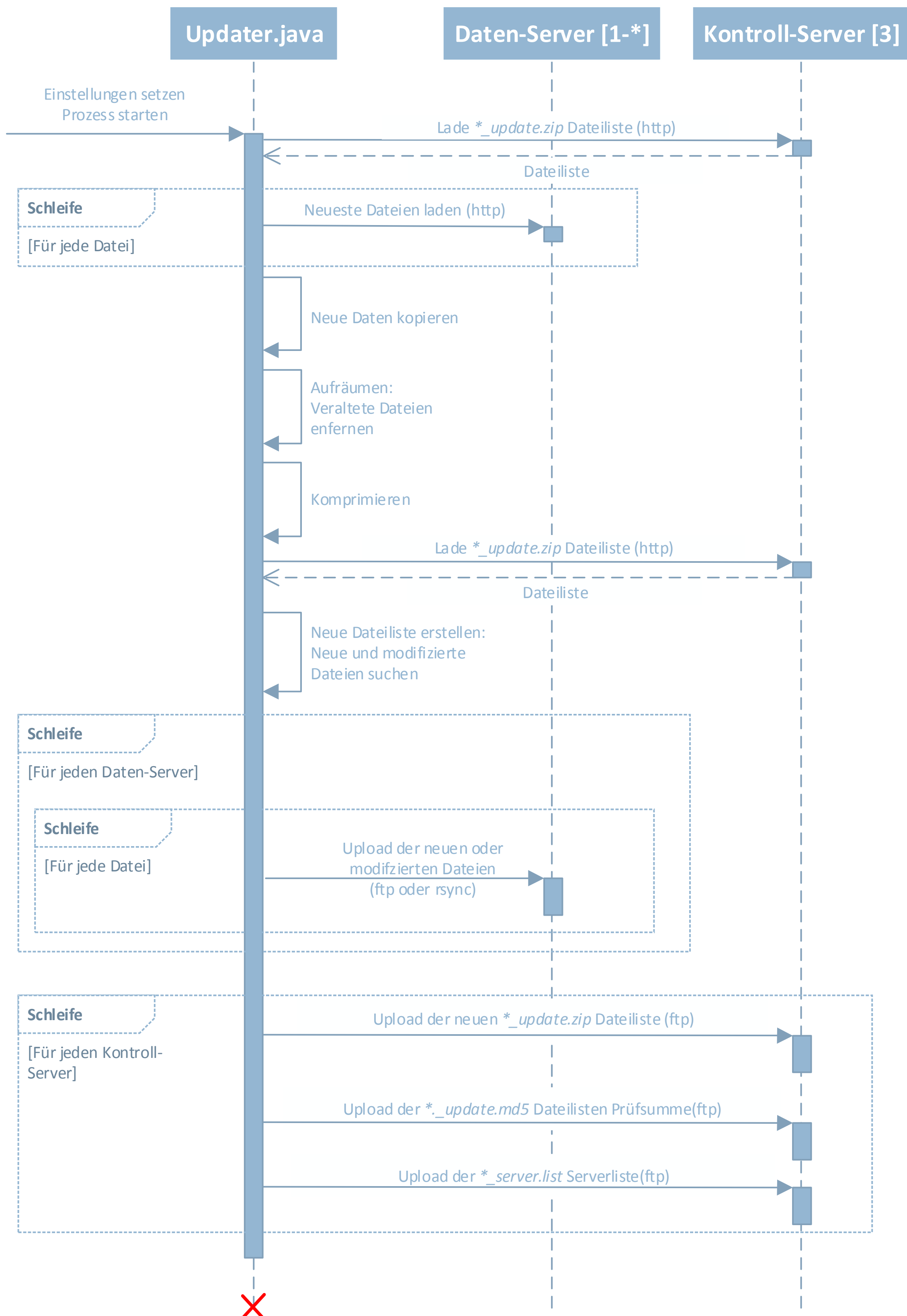

Abbildung 7: Prozessübersicht: Eine neue Version verteilen. [5]

5.2.3 Aktualisierungsanwendung

Nachdem eine neue Version durch die Verwaltungsanwendung auf die Kontroll- und Daten-Server geladen wurde, kann eine Aktualisierungssoftware[16] auf dem Rechner eines Endnutzers auf diese neue Version aktualisieren. In folgenden Fällen wird ein neuer Update-Prozess initiiert:

1. Automatisch nach dem Start von *JDownloader.jar*
2. Manuell durch einen Klick auf die „Nach Aktualisierungen suchen“ Schaltfläche in *JDownloader.jar* (oder andere Benutzerinteraktionen)
3. Manuell durch einen Start von *jdupdate.jar*

***.jar Archive:**

Eine **.jar*-Datei ist keine Anwendung im tatsächlichen Sinne. Vielmehr ist eine **.jar*-Datei eine Archivdatei, die, in der Regel, mehrere kompilierte Java Klassen (**.class* Dateien) und Ressourcen enthält. Eine solche **.jar*-Datei wird von einem Java Prozess (z.B. *java.exe*) geladen und ausgeführt. [6]

Eine laufende Anwendung kann sich auf Windows Systemen nicht selbst überschreiben. Dasselbe gilt für **.jar* Dateien sobald sie vom System Classloader der Java Virtual Machine (JVM) geöffnet wurden, da dieser den Schreibzugriff auf die Java Archive blockiert. Aus diesem Grund ist die Aktualisierungsanwendung in JDownloader in mehrere Anwendungen unterteilt, die sich, jeweils gegenseitig, aktualisieren und überschreiben können. Der genaue Aufbau der jeweiligen **.jar*-Archive kann der Build-Datei build.xml [6] entnommen werden.

- ***„JDownloader.jar“***
  ist der JDownloader Hauptprozess. Über diese **.jar*-Datei wird JDownloader üblicherweise gestartet. Lädt *JDownloader.jar* ein neues Update-Paket, so werden diese vorerst in einen Unterordner *„./update/“* des Stammverzeichnisses installiert.
- ***„tinyupdate.jar“***
  ist eine sehr kleine Anwendung. Sie wird vom *JDownloader.jar* Prozess gestartet, kurz bevor sich dieser selbst beendet. Die Aufgabe von *tinyupdate.jar* ist es, alle Dateien aus dem *„./update/“*-Unterordner ins Stammverzeichnis zu verschieben. Bereits vorhandene Dateien werden dabei überschrieben. Sollte *tinyupdate.jar* selbst aktualisiert worden sein, so wird sie von *JDownloader.jar* vor dem Start ins JDownloader Stammverzeichnis verschoben.

---

[16] Im JDownloader Entwicklerteam wird die Aktualisierungsanwendung meist als „Updateclient“ bezeichnet.

- ***„jdupdate.jar“***
  ist eine reine Aktualisierungsanwendung. Sollten wichtige Dateien für einen JDownloader Start fehlen, kann JDownloader über die *jdupdate.jar* auf den neuesten Stand gebracht werden. Da *jdupdate.jar* als eigener Prozess läuft, kann ein Update-Paket für *JDownloader.jar* geladen und installiert werden, ohne dabei den Prozess zu wechseln. Sie kombiniert dazu die Update-Funktionalität von *JDownloader.jar* und *tinyupdate.jar*. *Jdupdate.jar* hat, außer der JVM, keine weiteren Abhängigkeiten, und wird nur nach besonders gründlichen Tests aktualisiert. *Jdupdate.jar* fungiert damit als Notfall-Update-Routine.

5.2.3.1 Aktualisierungsprozess

Bei der folgenden Beschreibung des Update-Prozesses wird jegliche Interaktion zwischen Anwendung und Benutzer nicht aufgeführt. Tatsächlich wird der Nutzer über Dialoge gefragt ob er der Aktualisierung zustimmt, oder ob er JDownloader neu starten will. Falls der Benutzer eine solche Gelegenheit zum Abbruch wahrnimmt, wird der Update-Prozess an dieser Stelle einfach unterbrochen. Rollback-Routinen[17] gibt es im Update-System nicht. Der im Folgenden beschriebene Prozess ist in der Datei *jd.utils.WebUpdate.java* [7] sowie dem Paket *jd.update.** [8] zu finden.

Jeder Update-Prozess startet mit dem Aktualisieren der Daten-Serverliste. Dazu wird die neue Serverliste von einem Kontroll-Server geladen. Um die Last auf die drei Kontroll-Server gleichmäßig zu verteilen wird der in Kapitel 5.2.3.3.1 (Seite 35) beschriebene Algorithmus angewendet.

Der nächste Schritt lädt die MD5-Prüfsumme der aktuellen Dateiliste (*<Update-Zweig>_update.md5*). Falls diese Prüfsumme nicht mit der lokal gespeicherten Dateiliste übereinstimmt (Abbildung 8: opt(1)), wird die Dateiliste anschließend ebenfalls von den Kontroll-Servern geladen und lokal gespeichert. Die Dateiliste enthält die relativen Pfade aller Dateien und deren MD5-Prüfsumme. Diese Liste wird mit den lokalen Daten abgeglichen. Die daraus resultierende Liste (Abbildung 8: Liste(1)) enthält alle neuen oder modifizierten Dateien.

Anschließend prüft JDownloader.jar zuerst ob die Notfall-Aktualisierungsanwendung *jdupdate.jar* aktualisiert werden muss. Fall dem so ist, wird diese heruntergeladen und installiert.

Als nächstes werden alle Dateien in Liste(1) heruntergeladen und installiert. Dabei wird die Last, wie in Kapitel 5.2.3.3.2 (Seite 35) beschrieben, auf mehrere Daten-Server verteilt. Meistens können die Dateien nicht einfach überschrieben werden, weil der laufen-

[17] Routinen in der Anwendung, die, bei einer Unterbrechung der Installation eines Update-Pakets, alle bisher gemachten Änderungen rückgängig machen können.

de Prozess den Zugriff blockiert. Stattdessen speichert das Update-System die Dateien in einem temporären Unterordner „./update/".

Dabei gibt es einige Sonderfälle:

- **.extract* Archive. Manche Ordner wurden durch die Verwaltungsanwendung (Kapitel 5.2.2, Seite 26) zu Zip-Archiven komprimiert. Das reduziert die Dateianzahl und die Dateigröße und spart so Bandbreite und Zeit während der Übertragung. Diese Archive werden nach dem erfolgreichen Download sofort entpackt und anschließend gelöscht.
- *„/tools/tinyupdate.jar*" wird verwendet wenn JDownloader sich selbst überschreiben und damit neu starten muss. Neue Dateien werden dazu während des Aktualisierungsprozesses in einem Unterordner *„./update/*" gespeichert. Schlussendlich wird *tinyupdate.jar* gestartet und verschiebt diese Dateien in das JDownloader Stammverzeichnis sobald dieser beendet wurde. Da sich aber auch *tinyupdate.jar* im Zweifelsfall nicht selbst überschreiben kann, verschiebt *JDownloader.jar* diese Datei selbst an den endgültigen Pfad. Die beiden Prozesse laufen also möglichst nie gleichzeitig und überschreiben sich jeweils bei Bedarf gegenseitig.

Nachdem neue Dateien von den Daten-Servern geladen wurden, liegen diese im *„./update/*"-Unterordner. Um diese nun endgültig zu installieren muss JDownloader sich über die *tinyupdate.jar* Anwendung neu starten. Dieser Neustart wird über einen zusätzlichen „Helfer"-Prozess durchgeführt. Neben dem Verschieben aller Dateien von *„./udpate/*" nach *„./*" gehört es auch zu den Aufgaben von *tinyupdate.jar,* die *outdated.dat* Datei zu lesen und alle darin aufgeführten Dateien zu löschen.

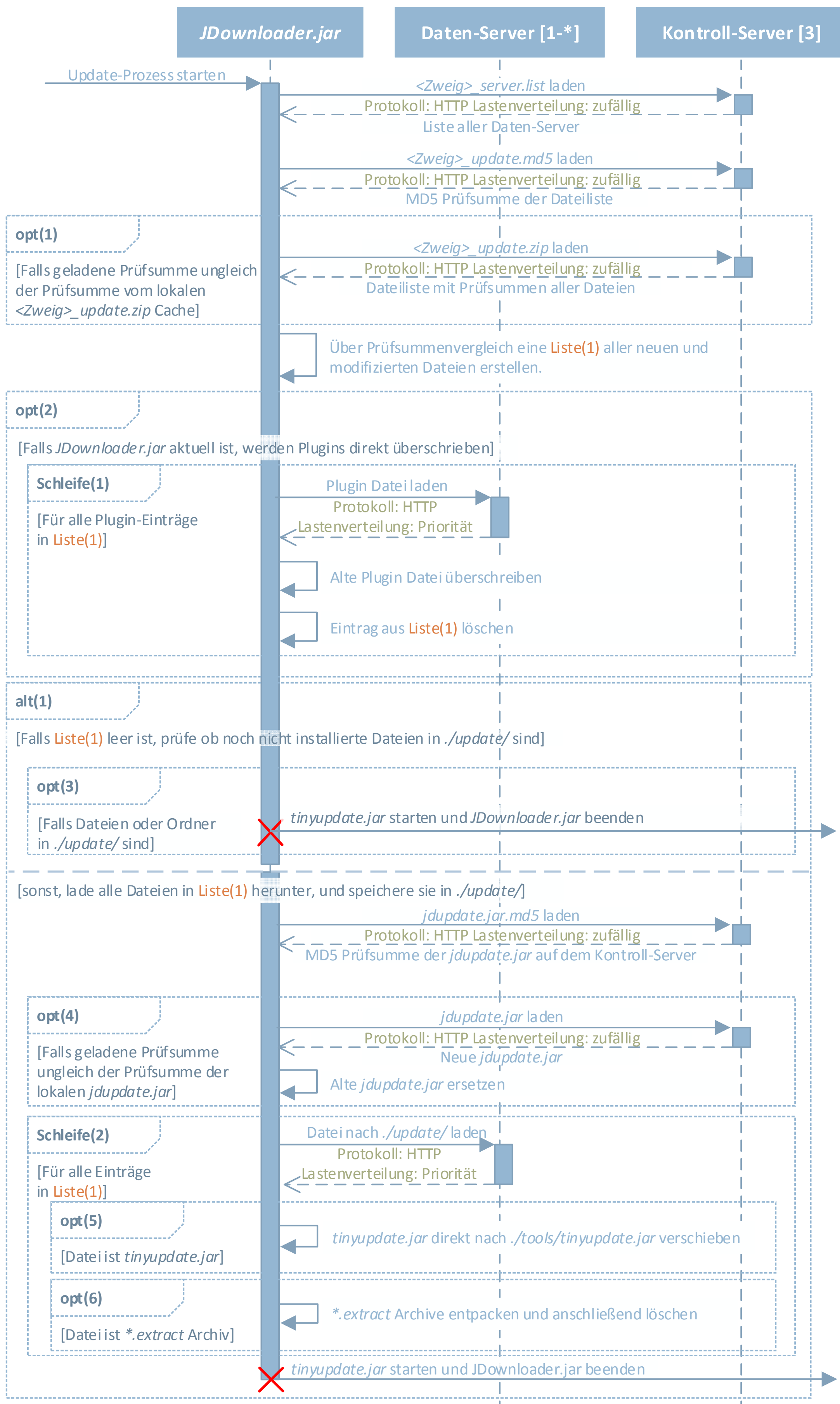

Abbildung 8: Aktualisierungsprozess (Teil 1) aus *JDownloader.jar* gestartet

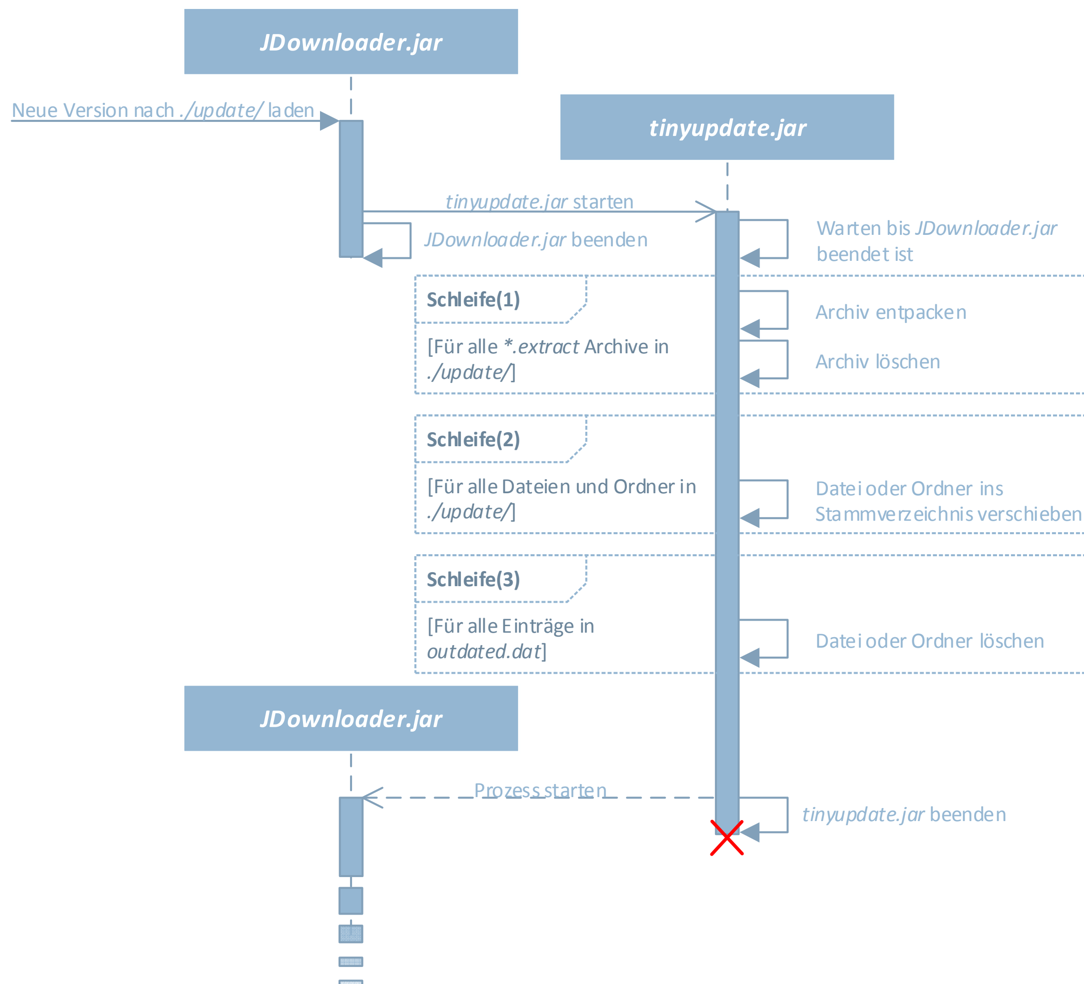

Abbildung 9: Aktualisierungsprozess (Teil 2) aus *JDownloader.jar* gestartet

5.2.3.2 Plugins zur Laufzeit installieren

Enthält Liste(1) Plugin-Dateien, werden diese direkt geladen und installiert, falls *JDownloader.jar* selbst aktuell ist. Ein Neustart von *JDownloader.jar* über *tinyupdate.jar* ist dann nicht nötig. Falls *JDownloader.jar* jedoch selbst aktualisiert werden muss könnte sich die Plugin-Schnittstelle geändert haben. Aus diesem Grund wird dieser Schritt in einem solchen Fall übersprungen. Plugin-Dateien sind nicht in **.Jar*-Archiven verpackt und werden deswegen von der JVM nicht blockiert. Sie können zur Laufzeit überschrieben werden. JDownloader 0.9581 kann bereits geladene Plugins zur Laufzeit allerdings nicht austauschen. Der Endnutzer kann die aktualisierten Plugins also meist erst nach einem Neustart nutzen.

5.2.3.3 Lastverteilung

Die Anzahl der Anfragen und die benötigte Bandbreite sind zu hoch für einen einzelnen Server (vgl. Kapitel 5.2.1, Seite 23). Aus diesem Grund werden die Anfragen der Aktualisierungsanwendung auf mehrere Server verteilt. Die zu Grunde liegende Logik bei Kontroll- und Daten-Servern ist leicht unterschiedlich.

5.2.3.3.1 Lastverteilung der Kontroll-Server

Um die Last auf die drei Kontroll-Server gleichmäßig zu verteilen wird die Liste der Kontroll-Server einmalig zufällig gemischt. Anschließend werden höchstens 10 Versuche unternommen über die gewünschte Anfrage eine Antwort vom Kontroll-Server zu erhalten. Nach dem 10. Fehlschlag bricht der Prozess mit einer Fehlermeldung ab. Diese Art der Lastenverteilung wird für alle Zugriffe auf die Kontroll-Server genutzt.

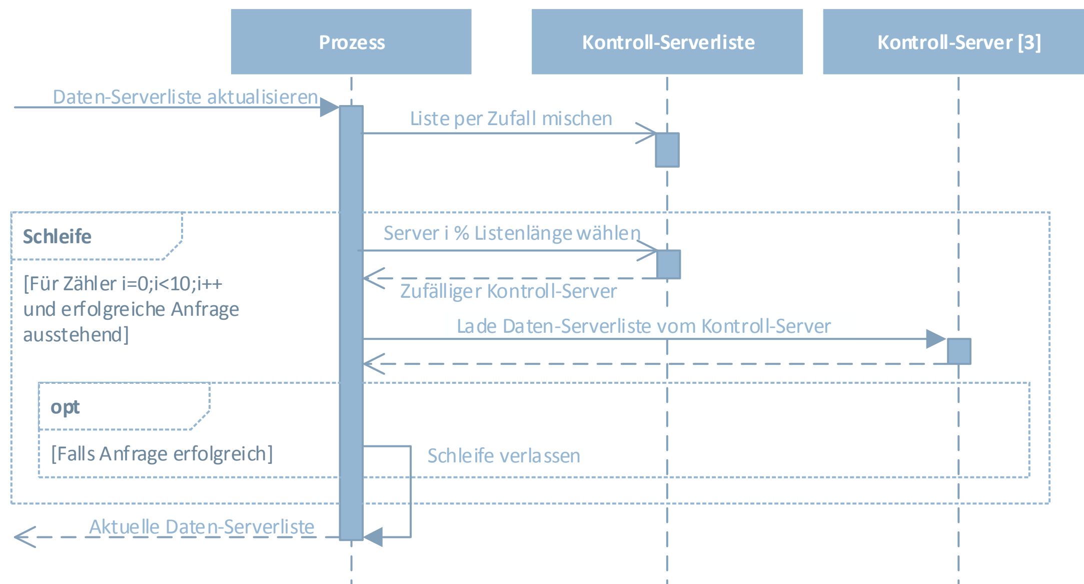

Abbildung 10: Lastenverteilung der Kontroll-Server[18]

5.2.3.3.2 Lastverteilung der Daten-Server

Der Download einer Datei von einem Daten-Server ist in Abbildung 11 illustriert. Zuerst wird ein Daten-Server ausgewählt.

Zur Lastverteilung werden dazu entweder die Prioritätsfaktoren aus der Serverliste oder die Verbindungszeiten zu den Servern genutzt. Im Falle der Prioritätsfaktoren wird die Last gemäß den Prioritäten verteilt (Abbildung 6, Seite 26). Ist in der Serverliste keine Priorität angegeben, so wird bevorzugt der Server mit der geringsten durchschnittlichen Verbindungszeit verwendet. Dieser Ansatz beruht auf der Annahme, dass ein überlasteter Server schlechte Verbindungszeiten liefert. Zunächst wird dazu jeder Daten-Server mindestens einmal verwendet. Anschließend wird von dem Daten-Server, der die kürzesten Verbindungszeiten liefert geladen. Die durchschnittlichen Verbindungszeiten werden durch jede aufgebaute Verbindung aktualisiert.

[18] Methode jd.update.WebUpdater.updateAvailableServers()

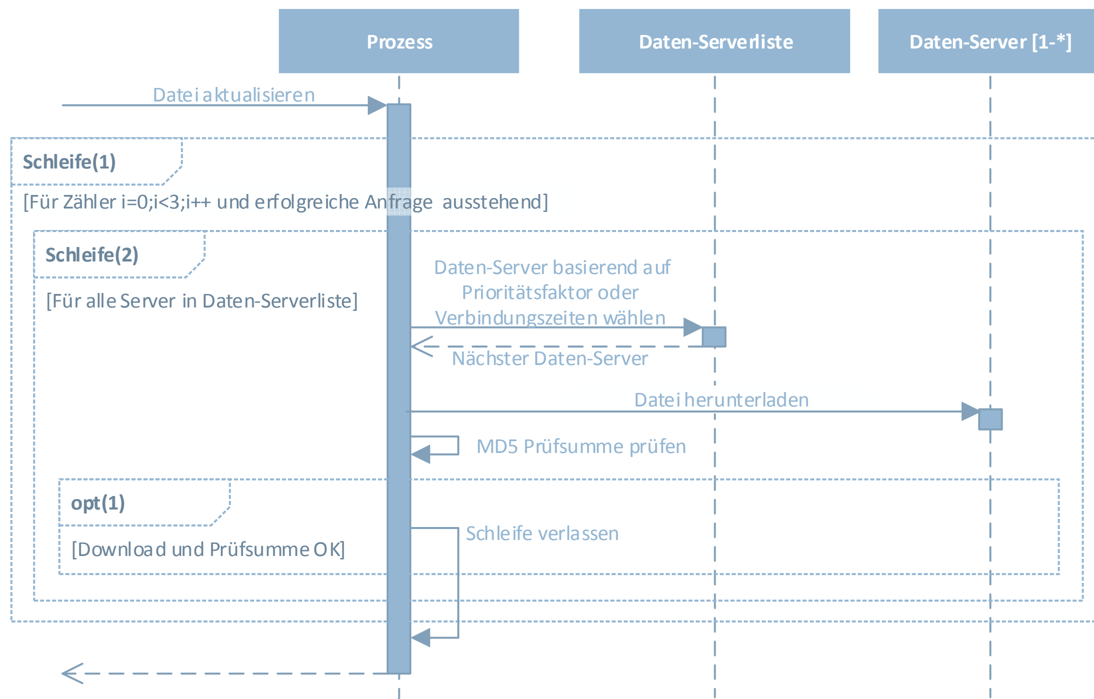

Abbildung 11: Lastverteilung der Daten-Server[19]

Nach der Wahl des nächsten Daten-Servers wird die Datei von diesem heruntergeladen und in einer temporären Datei gespeichert. Stimmt die Prüfsumme der geladenen Datei mit der Angabe aus der Dateiliste nicht überein, wird die Datei gelöscht. Andernfalls wurde die Datei korrekt geladen und wird übernommen.

5.2.3.4 Update-Zweige

Laut Anforderung 5.1.3 (Seite 19) wurde eine Möglichkeit gefordert verschiedene Update-Zweige zu verwalten. Diese Funktion wurde über unterschiedliche Namensräume realisiert. Auf den Kontroll-Servern kann beispielsweise die Dateiliste des „NIGHTLY"-Zweigs unter *http://update1.jdownloader.org/NIGHTLY_update.zip* geladen werden. Die Dateien selbst liegen auf den Daten-Servern jeweils in einem Unterordner der nach dem Zweignamen benannt ist.

Um zwischen verschiedenen Zweigen zu wechseln, muss der Endnutzer *JDownloader.jar* oder *jdupdate.jar* mit einem Zusatzparameter starten:

```
>java -jar JDownloader.jar -branch <Zweigname>
```

Mit diesem Zusatzparameter wechselt der Zweig, und damit die HTTP-Adressen, unter denen die Aktualisierungsanwendung auf die Kontroll- und Daten-Server zugreift.

[19] jd.update.FileUpdate.update(…)

5.2.3.5 Betriebssystem Filter

Um Anforderung 5.1.12 (Seite 21) zu erfüllen, wurde ein einfaches Betriebssystem Filter implementiert. Nötig ist dies, um bestimmte Dateien nur an das passende Betriebssystem auszuliefern. Ein Windows System hätte beispielsweise keine Verwendung für ein Unix Shell-Script. Das Filter basierte auf der Auswertung des relativen Dateipfades in der Dateiliste. Beim Einlesen und Auswerten der Dateiliste[20] wird diese anhand folgender Regeln gefiltert:

- Enthält der Pfad „windows" wird die Datei nur auf Windows Rechnern geladen und installiert.
- Enthält der Pfad „mac" wird die Datei von Mac OS Systemen akzeptiert
- Enthält der Pfad „linux" wird die Datei weder auf Windows, noch auf Mac OS Systemen installiert, aber auf allen anderen.

	Windows System	Mac System	Andere Systeme
Pfad enthält „windows"	installieren	verwerfen	verwerfen
Pfad enthält „mac"	verwerfen	installieren	verwerfen
Pfad enthält „linux"	verwerfen	verwerfen	installieren

Tabelle 3: Entscheidungstabelle Betriebssystem Filter

5.2.3.6 Fehlerkorrektur und Fallback Routinen

Bei den bisherigen Beschreibungen wurde immer davon ausgegangen, dass die Aktualisierung erfolgreich lief. Eine gesonderte Betrachtung jedes möglichen Fehlers ist nicht nötig, weil die Aktualisierungsanwendung jegliche Fehler gleich behandelt. Im Falle eines Fehlers bricht das System einfach mit einer entsprechenden Meldung ab. Alle bis dahin geladenen und installierten Dateien werden behalten. Dadurch kann es zu inkonsistenten Zuständen kommen. Wird beispielsweise JDownloader.jar aktualisiert, aber einige benötigte Bibliotheken können nicht erneuert werden, kann es durch fehlerhafte Schnittstellen zu Laufzeitfehlern kommen.

Im schlimmsten Fall wird dadurch das Update-System selbst beschädigt. JDownloader könnte sich in so einem Fall nicht mehr selbst aktualisieren. Das Update-System besitzt keinen zuverlässigen Schutz vor solchen Situationen. Für den Notfall existiert jedoch eine Datei „*jdupdate.jar*". Sie beinhaltet die komplette Update-Logik und kann sich ebenfalls selbst aktualisieren. Startet der Endnutzer die „*jdupdate.jar*", bringt diese seine JDownloader Installation auf den neusten Stand. Während eine neue JDownloader Version vor der Veröffentlichung keine aufwendigen Tests durchläuft, wird die

[20] jd.update.WebUpdater.parseFileList(…) Zeile 438

jdupdate.jar intensiv getestet. Vor allem die Update-Routinen werden überprüft, um einen fehlerfreien Ablauf zu garantieren. Neben der *jdupdate.jar,* die der Nutzer im Zweifelsfall selbst starten muss, wurde ein weiteres System eingerichtet. Das „Dynamics" System ist ein Plugin-System das, sehr früh nach dem Start der JDownloader Anwendung, Java Klassen (Dynamic Plugins) von den Update-Servern nachladen und ausführen kann. Über diese Dynamic Plugins könnten Installationen die sich selbst in eine „Sackgasse" aktualisiert hatten wieder an das System angebunden werden. Zum Herunterladen werden jedoch ebenfalls die Routinen der Aktualisierungsanwendung verwendet.

6 Evaluation

Schon bei der Analyse der Realisierung fällt auf, dass nicht alle Anforderungen aus Kapitel 5.1 erfüllt wurden. Inzwischen ist das System seit vier Jahren im Produktiveinsatz. Das Nachfolge-System befindet sich derzeit im Beta-Test.

6.1.1 Erfüllung der Anforderungen

Um zu erkennen wo das Update-System an seine Grenzen stößt, und warum eine neue Version entwickelt wurde, sollen die Anforderungen rückblickend aufgearbeitet werden.

Anforderung	erfüllt
„Sackgassen vermeiden“	Nein
Inkonsistente Zustände vermeiden	Nein
Update-Zweige	Ja
Kurze Reaktionszeiten	teilweise
Verwaltungsanwendung	Ja
Serveranforderungen	Ja
Skalierbarkeit	Ja
Inkrementelle Aktualisierung	Ja
Plugins zur Laufzeit aktualisieren	teilweise
Konnektivität durch HTTP	Ja
Plattformunabhängigkeit	Ja
Betriebssystemfilter	teilweise
Datei- und Ordneranzahl	teilweise

Tabelle 4: Übersicht über die Anforderungserfüllung

6.1.1.1 „Sackgassen“ werden nicht zuverlässig vermieden

Probleme und Fehler in der Umsetzung der Anforderung 5.1.1:

- **Anforderung wurde nicht beachtet**
- **Reparatursystem „Dynamic Plugins“ ist mangelhaft umgesetzt**
- **Reparatursystem „jdupdate.jar“ funktioniert nicht automatisch**

Diese Anforderung wurde nicht erfüllt. Fehler in JDownloader Komponenten oder genutzten Bibliotheken führen schnell zu einem Ausfall des kompletten Update-Systems.

Beispiel:

> Ein Fehler in der HTTP-Bibliothek führt dazu, dass keine Proxy-Verbindungen mehr genutzt werden können. Nutzer einer Proxy-Verbindung sind vom Update-System abgeschnitten sobald sie auf diese fehlerhafte Version aktualisieren. Es gibt keine Routinen, die ein solches Problem vor einer Aktualisierung erkennen könnten.

Nachdem solche Fehler einige Male passiert waren, ging man dazu über, neue Versionen zuerst in einem neuen Zweig zu testen bevor man sie im NIGHTLY oder gar STABLE Zweig veröffentlichte. Diese Vorsichtsmaßnahme verlangsamte den ganzen Prozess erheblich.

Das System unternimmt keine Maßnahmen zur Vermeidung solcher Sackgassen. Allerdings wurden mit den „Dynamic Plugins“ und *jdupdate.jar* zwei Reparatursysteme implementiert über die eine Installation in der Sackgasse wieder an das Update-System angebunden werden könnte. Beide haben jedoch deutliche Mängel:

„Dynamic Plugins“ werden kurz nach dem JDownloader Start direkt von einem Daten-Server geladen und ausgeführt. Über sie können Hotfixes[21], aber auch Code der ein beschädigtes Update-System reparieren kann, eingespielt werden. Da die „Dynamic Plugins“ zum Laden der Plugins aber die Routinen des Update-Systems verwenden, ist die Chance, dass dieses Reparatursystem bei einem Ausfall der Aktualisierungsanwendung ebenfalls ausfällt, sehr groß. Abgesehen davon können „Dynamic Plugins“ nichts ausrichten, falls JDownloader nicht gestartet werden kann.

Falls JDownloader gar nicht mehr gestartet werden kann, muss der Nutzer den *jdupdate.jar* Prozess starten. Dieser kann die Installation dann wieder reparieren. Allerdings muss dazu der Benutzer selbst aktiv werden. In vielen Fällen wirkt sich ein Ausfall der Aktualisierungsanwendung so aus, dass der Benutzer keine Benachrichtigungen über neue Aktualisierungen erhält. Die Motivation den *jdupdate.jar* Prozess ohne offensichtlichen Grund zu starten ist sehr gering. So kommt es häufig vor, dass ein Nutzer vom Update-System abgeschnitten bleibt, obwohl ein einfacher Start von jdupdate.jar viele Probleme beheben würde.

[21] Hotfix: Eine eilige Aktualisierung, die in der Regel einen kritischen Fehler behebt

6.1.1.2 Inkonsistente Zustände können auftreten

Probleme und Fehler in der Umsetzung der Anforderung 5.1.2:

- **Fehlende Unterstützung von „Versionen“**
- **Aktualisierungen können teilweise installiert werden**
- **Das Löschen von Dateien ist über die *outdated.dat* Datei nur ungenügend gelöst**

Das Update-System kennt keine „Versionen“, sondern nur eine Dateiliste mit MD5-Prüfsummen. Es kann zwar zwischen den Zuständen „aktuell“ und „nicht aktuell“ unterscheiden, aber kann nicht auf eine ältere Version zurückgehen. Eine teilweise erfolgreiche Aktualisierung kann damit nur ausgebessert werden, in dem komplett auf die neuste Version aktualisiert wird. Durch Fehler im Update-Prozess, wie beispielwiese Verbindungsabbrüche oder der Absturz des Computers, kann es sehr einfach zu inkonsistenten Datenzuständen kommen.

Im Weiteren war gefordert, dass das Update-System nicht nur neue und modifizierte Dateien installieren kann, sondern auch nicht mehr benötigte Dateien löschen kann. Diesbezüglich ist der Ansatz über die Dateiliste ungeeignet. Schließlich enthält diese Liste ausschließlich Dateien die es noch gibt. Um dennoch alte Dateien löschen zu können nutzt man eine Datei „*outdated.dat*“, die alle jemals existierenden und nicht mehr benötigten Dateien, enthält. Einträge können erst entfernt werden, wenn man sich sicher sein kann, dass diese Datei auf keinem Endnutzerrechner mehr zu finden ist. Diese Sicherheit ist nie gegeben, und so wurde die Datei im Laufe der Jahre immer größer. Da das komplette System keine „Versionen“ im klassischen Sinne kennt, gibt es auch keine Aufzeichnungen über alte Datenbestände. Eine automatische Generierung dieser *outdated.dat*-Datei ist deshalb nicht möglich. Die Entwickler müssen die Datei folglich manuell pflegen und erweitern. Diese Verwaltung der zu löschenden Dateien ist sehr fehleranfällig. In der Praxis verbleiben deswegen viele ungenutzte Dateien auf den Rechnern der Nutzer. Diese „Dateileichen“ können Fehler verursachen, verbrauchen aber in jedem Fall unnötig Speicherplatz.

6.1.1.3 Kurze Reaktionszeiten werden nur bedingt erreicht

Probleme und Fehler in der Umsetzung der Anforderung 5.1.4:

- **Hohe Last auf den Update-Servern erschwert das Veröffentlichen neuer Versionen**
- **Daten- und Kontroll-Server können nicht kurzfristig entlastet werden**

Theoretisch dauert es nur wenige Minuten bis eine neue Version veröffentlich ist. In der Praxis kommt es häufig vor, dass kurz nach der Veröffentlichung einer neuen Version

erste Fehlermeldungen durch die Benutzer eingehen. In solchen Fällen muss oft sehr kurzfristig ein Fehler behoben werden.

Leider sind die Update-Server kurz nach der Veröffentlichung einer neuen Version stark ausgelastet. Die Auslastung steigt dabei üblicherweise sprunghaft auf Vollauslastung an. Ein Server unter Vollauslastung kann Anfragen gar nicht, oder nur sehr langsam, beantworten. Die Kommunikation mit der Verwaltungsanwendung ist damit ebenfalls gestört.

Das führt dazu, dass es fast unmöglich wird in den ersten Stunden nach der Veröffentlichung einer neuen Version erneut eine neue Version zu veröffentlichen. Der einzige Weg dies zu tun, ist die Last auf die Server zu verringern. Dazu müssen die Daten-Serverlisten offline[22] genommen werden. Das führt schließlich zu einem langsamen Rückgang der Last auf die Daten-Server. Eine schnelle Entlastung der Daten-Server ist nicht möglich. Selbst nach der Modifikation der Serverlisten muss man warten bis alle aktiven Update-Prozesse beendet wurden. Bei einer Überlastung der Server kann dies mehrere Stunden dauern. Die Kontroll-Server können zudem gar nicht entlastet werden.

Die benötigte Zeit um eine neue Version zu veröffentlichen kann also nur dann klein gehalten werden, wenn die Daten-Server nicht voll ausgelastet sind.

6.1.1.4 Plugin-Aktualisierungen zur Laufzeit

Probleme und Fehler in der Umsetzung der Anforderung 5.1.4 :

- **Plugins werden zwar zur Laufzeit aktualisiert, aber nicht neu geladen**
- **Automatische Suche nach Aktualisierungen nur beim Start**

Die Aktualisierungsanwendung sucht nach dem Starten von JDownloader einmalig nach neuen Dateien und Plugins. Falls Benutzer JDownloader sehr lange am Stück laufen lassen, bekommen sie keine Aktualisierungsbenachrichtigung solange sie JDownloader nicht neu starten, oder per Klick auf die „Nach Aktualisierungen suchen“ Schaltfläche eine erneute Suche starten.

Falls neue Versionen von Plugins geladen werden können, werden diese zur Laufzeit wie gefordert installiert. Die meisten Plugins werden jedoch einmalig beim Programmstart in den Arbeitsspeicher geladen und können dort nur durch einen Neustart der ganzen Anwendung ersetzt werden. Es sei an dieser Stelle darauf hingewiesen, dass dies kein reines Problem des Update-Systems ist. Vielmehr müsste diese Anforderung von Plugin- und Update-System gemeinsam erfüllt werden. Das Plugin-System in JDownloader 0.9851 ist jedoch nicht fähig Plugins zur Laufzeit zu ersetzen.

[22] In der Praxis werden die Dateinamen, unter denen die Listen abgerufen werden können, geändert. Anfragen werden dann vom HTTP-Dienst mit „File not Found“ beantwortet.

6.1.1.5 Betriebssystem Filter

Probleme und Fehler in der Umsetzung der Anforderung 5.1.12:

- **Das Filter verursacht viele Fehler durch unbeabsichtigte Marker im Pfad**

Das geforderte Filter für Dateien, die nur an bestimmte Betriebssysteme ausgeliefert werden dürfen, wurde über Marker realisiert. Wird einer der Marker „windows", „linux" oder „mac" im relativen Pfad einer Datei gefunden, wird die entsprechende Datei auch nur für die genannten Betriebssysteme installiert. Man muss also nur darauf achten, dass beispielweise Dateien die nur für Windows gedacht sind, in einem Unterordner namens „*[…]/windows/[…]*" liegen.

In der Praxis führt das aber zu Problemen sobald solche Marker unbeabsichtigt im Pfad einer Datei landen. Zur Illustration soll ein Beispiel dienen:

Beispiel:

> Nehmen wir an, ein Entwickler erstellt ein neues Plugin für die hypothetische Seite „filemachine.com", um von dort Dateien laden zu können. Die entsprechende Plugin-Datei würde *FileMachineCom.class* heißen. Durch das Betriebssystemfilter würde dieses Plugin nur an Mac OS Systeme verteilt werden, da im relativen Dateipfad *jd/plugins/hoster/File**Mac**hineCom.class* der Marker „mac" steckt.

Der Entwickler muss also bei der Namensgebung seiner Klassen und Komponenten darauf achten, keine dieser Marker zu verwenden. Dies gelingt in der Praxis häufig nicht. Wird der Fehler bemerkt, muss die Datei umbenannt werden. Die bereits verteilte Datei muss aber ebenfalls wieder entfernt werden. Die Problematik beim Löschen von Dateien durch das Update-System wird in Kapitel 6.1.1.2 (Seite 41) näher betrachtet.

6.1.1.6 Dateianzahl und Speicherbedarf

Probleme und Fehler in der Umsetzung der Anforderung 5.1.13 :

- **Die reine Dateianzahl ist nicht ausreichend um die maximale Leistungsgrenze zu definieren**
- **Unterschiedliche Verbindungsgeschwindigkeiten wurden bei der Analyse der Anforderung ignoriert**
- **Das starke Wachstum wurde bei der Anforderungsanalyse nicht beachtet**

Die hohe Auslastung der Daten-Server liegt unter anderem an den vielen Anfragen der Aktualisierungsanwendungen beim Herunterladen neuer Dateien. In Kapitel 5.2.1 (Seite

23) wurde anhand einer Beispielrechnung bereits dargestellt, dass die Server sehr viele Verbindungen in Kombination mit einer hohen Bandbreitenauslastung zu verkraften haben. Das Update-System kennt theoretisch kein Limit in der Anzahl der Dateien. Praktisch steigt die Zeit, die für das Aktualisieren auf eine neue Version, aber auch für das Bereitstellen dieser, benötigt wird, mit der Anzahl der geänderten Dateien. Die benötigte Zeit für den Download einer Datei $T_{Download}$ und für die anschließende Prüfsummenberechnung T_{MD5} ergeben, multipliziert mit der Anzahl aller geänderten Dateien D_{neu}, die gesamte Installationszeit $T_{Installation}$.

$$T_{Installation} = D_{neu} \cdot (T_{Download} + T_{MD5})$$

JDownloader besteht aus vielen kleinen (im Durchschnitt ~30kb) Dateien. Die Zeit für den Download wird deswegen deutlich von der Zeit $T_{Verbindungsaufbau}$ beeinflusst die zum Aufbauen der Verbindung selbst benötigt wird.

$$T_{Installation} = D_{neu} \cdot (T_{Verbindungsaufbau} + T_{Datentransfer} + T_{MD5})$$

$T_{Verbindungsaufbau}$ ist unter anderem erheblich von der Internetverbindung, dem Routing zum gewählten Daten-Server, der Auslastung aller beteiligten Netzkomponenten, aber auch von Anwendungen auf dem Rechner des Endkunden, wie z.B. Firewalls, abhängig. Schwankungen zwischen wenigen Millisekunden und einigen Sekunden sind durchaus üblich.

Beispiel:

Es wurde die durchschnittliche Zeit zum Aufbau einer HTTP-Anfrage an den Update-Server update1.jdownloader.org gemessen. Die Messungen wurden annähernd zeitgleich bei gleichbleibender Serverauslastung getätigt. Der Serverstandort ist Nürnberg in Deutschland. Die Unterschiede sind auf die langsame Internetverbindung des Nutzers in Australien, aber vor allem auf das Routing nach Nürnberg zurückzuführen.

Endnutzer Rechner und Anbindung	$\emptyset T_{Verbindungsaufbau}$
Fürth, Deutschland über Telekom VDSL 50 mbit	53 ms
Perth, Australien über Amnet ADSL 3 mbit	1281 ms

Tabelle 5: Zeit für den Aufbau einer HTTP-Download Anfrage an update1.jdownloader.org

$T_{Datentransfer}$ ist hauptsächlich von der Internetanbindung des Nutzers und der Auslastung des Servers abhängig.

T_{MD5} ist die Zeit, die benötigt wird, um die MD5-Prüfsumme einer Datei zu errechnen. Sie hängt hauptsächlich von der Dateigröße selbst und der Geschwindigkeit des lokalen Dateisystems beim Endnutzer ab.

Da vor dem Herunterladen und Installieren zuerst die Dateiliste mit allen Dateien über die MD5-Prüfsumme verglichen werden, kommt zur Installationszeit noch T_{MD5} multipliziert mit der Anzahl aller Dateien D_{alle}hinzu. Zusammen ergibt sich die Zeit für die Aktualisierung von JDownloader T_{Update} aus:

$$T_{Update} = D_{neu} \cdot \left(T_{Verbindungsaufbau} + T_{Datentransfer} + T_{MD5}\right) + D_{alle} \cdot T_{MD5}$$

Das Update-System kann die geforderte Menge an Dateien und Daten bei einer schnellen Internetanbindung zwar in akzeptabler Zeit aktualisieren, benötigt aber unverhältnismäßig viel Zeit wenn die Aktualisierung über eine langsame Internetverbindung läuft.

Es zeigt sich also, dass die Anforderung zwar durchaus legitim ist, aber nicht ausreichend ist um die Leistungsgrenzen des Update-Systems zu definieren. Die Anforderung 5.1.13 befasst sich zudem nicht mit dem weiteren Wachstum von JDownloader. Dateianzahl und Datenmenge sind in den vergangenen Jahren erheblich gewachsen. Diese und andere Anforderungen müssen folglich überarbeitet werden.

6.1.2 Neue und geänderte Anforderungen

Aus den bisher gewonnen Erfahrungen, aber auch aus neuen Voraussetzungen ergeben sich neue Anforderungen und neue Sichtweisen auf die alten. Ausgehend von dem hier vorgestellten Update-System wurden die Anforderungen aus heutiger Sicht betrachtet. Im Oktober 2012 wurde die Planung für ein neues Update-System, das Update-System 2, begonnen. Im Unterschied zum ersten System wurde das neue System von Anfang an für eine kommerzielle Nutzung und Lizensierung ausgelegt. Die Anforderungen setzen sich also aus Anforderungen an das JDownloader Projekt und Anforderungen, die ein potentieller Kunde stellen könnte, zusammen. Die Anforderungen aus Kapitel 5.1 werden unverändert übernommen, sofern sie im Anschluss nicht erneut aufgeführt sind.

6.1.2.1 Vollständig automatisierbarer Verteilprozess

Eines der Kernziele des neuen Systems ist die Eingliederung des Update-Systems in die Prozesskette der „Kontinuierlichen Integration“. Die Zyklen von einer Fehlermeldung bis zur Veröffentlichung des korrigierten Codes sollen so kurz wie möglich werden. Um dieses Ziel zu erreichen ist es notwendig neue Versionen nach dem Build- und Testprozess vollständig automatisch zu verteilen. Eine Kommandozeilenanwendung soll die Ankopplung des Update-Systems an das automatische Build-System vereinfachen.

6.1.2.2 Ausgereifte Verwaltungsanwendung

Neben einer Kommandozeilenanwendung muss eine Anwendung mit Benutzeroberfläche umgesetzt werden. Eine einfache Bedienoberfläche soll Fehler beim Veröffentlichen neuer Pakete vermeiden und Zugriff auf alle Funktionen zum Verwalten von verschiedenen Versionen und Zweigen enthalten:

- Aktuellen Datenbestand herunterladen
- Benutzergruppen verwalten und zuweisen
- Neues Update-Paket veröffentlichen

6.1.2.3 Bessere Lastkontrolle

Die Probleme im Update-System 1 haben gezeigt, dass es auch unter Volllast jederzeit möglich sein muss, neue Update-Pakete zu veröffentlichen. Um das zu gewährleisten, muss es möglich sein, die Update-Server jederzeit, möglichst schnell, zu entlasten.

6.1.2.4 Benutzergruppen statt Update-Zweige

Das Prinzip von verschiedenen Update-Zweigen hat sich bewährt. Allerdings ist es sehr aufwändig eine Version von einem Zweig auch für andere Zweige freizugeben. Das Veröffentlichen einer Version unter verschiedenen Zweigen ist ebenfalls nur möglich, indem die Daten dupliziert wurden. Der Ansatz über Namensräume auf den Daten-Servern zu arbeiten hat sich hier also weniger bewährt. Das System der Update-Zweige soll deswegen fallen gelassen werden. Stattdessen sollen Benutzer-Gruppen eingeführt werden. Über die Verwaltungsanwendung muss es möglich sein, eine bestimmte Version nur für eine bestimmte Benutzergruppe zu veröffentlichen.

Beispiel:

> Eine neue Version wird veröffentlicht, und zunächst nur für einen kleinen Kreis der Gruppe „Tester“ freigegeben. Nach erfolgreichen Tests kann diese Version schließlich für alle veröffentlicht werden.

Das Ändern der Zielgruppe einer Version sollte nicht den kompletten Prozess des Veröffentlichens erneut durchlaufen, wie es im Update-System 1 der Fall ist.

Um dennoch echte „Update-Zweige“ zu unterstützen soll das System beliebig viele Projekte verwalten können. Möglich wäre beispielsweise ein alternativer Zweig für eine Kommandozeilenvariante des JDownloaders.

6.1.2.5 Sicherheit durch Signaturen

Die Notwendigkeit von Signaturen wurde bereits in Kapitel 4.4 auf Seite 16 erläutert. Das erste Update-System hatte diese wichtige Funktionalität aber nicht umgesetzt. Im

neuen System sollen alle Update-Pakete durch die Verwaltungsanwendung beim Veröffentlichen signiert werden.

6.1.2.6 Optimieren auf viele kleine Update-Pakete

Der Ansatz der „Kontinuierlichen Integration“ fordert viele kleine Aktualisierungen. Im JDownloader Projekt werden Plugins deutlich häufiger aktualisiert als andere Programmteile. Der Overhead um beispielsweise ein einziges Plugin zu aktualisieren sollte möglichst gering sein.

Beispiel:

Die durchschnittliche Dateigröße einer Plugin-Datei ist etwa 10kb. Um ein Update-Paket dieser Größe zu laden, müsste das alte System zuerst die etwa 90kb große Dateiliste laden, um die geänderte Plugin-Datei zu erkennen, und anschließend die Plugin-Datei von einem Daten-Server laden. Die Nutzdatenrate liegt folglich bei $\frac{10}{10+90} = 10\%$.

Der Ansatz, über Dateilisten zu arbeiten, sorgt für eine schlechte Nutzdatenrate. Je kleiner die Update-Pakete dabei werden, desto schlechter wird die Nutzdatenrate.

6.1.2.7 Serveranforderungen

Die Anforderung serverseitig nur Standardsoftware nutzen zu dürfen (Kapitel 5.1.6, Seite 20) entfällt. JDownloader wird inzwischen kommerziell vermarktet und das Projekt ist nicht mehr länger auf günstige Hosting-Angebote oder gar gespendete Server angewiesen.

6.1.2.8 Leistungsfähigkeit

Für das Update-System 1 wurde die maximale Anzahl an Dateien und die Datenmenge als Referenz verwendet um die Mindestleistung des Update-Systems zu definieren. Wie sich herausstellte, reicht dies aber bei weitem nicht aus. Für das neue System wurden neben der Dateianzahl weitere Angaben verwendet.

Die maximale Dateianzahl wurde auf 10000 Dateien, verteilt auf maximal 500 Megabyte an Daten erhöht.

Der Einfluss von Verbindungszeiten (vgl. Kapitel 6.1.1.6, Seite 43) muss minimiert werden. Aufgrund der stark gestiegenen Dateianzahl wird man den Ansatz „Eine Anfrage pro Datei“ fallen lassen müssen.

Das System muss mindestens 1000000 gleichzeitige Installationen, von denen jede mindestens einmal pro Stunde nach einer neuen Version sucht, und diese bei Bedarf installiert, verwalten können.

6.1.2.9 Erweiterte Inkrementelle Update-Pakete

Das Konzept von Update-Paketen, die nur die Änderungen von einen auf den nächsten Stand enthalten, soll ausgebaut werden. Während im Update-System 1 dateibasierend gearbeitet wurde, sollen im neuen System auch Archive und andere größere Dateien behandelt werden. Zunächst reicht ein inkrementelles Aktualisieren von **.jar*-Archiven aus. Java Anwendungen werden üblicherweise in **.jar*-Archive verpackt. Dateien in den Archiven die sich nicht geändert haben, sollen nicht übertragen werden.

6.1.2.10 Kompression

Um Bandbreite und Transfervolumen während der Übertragung der Update-Pakete zu sparen, müssen diese komprimierbar sein. Als Kompressionsgsalgorithmus wird LZMA[23] empfohlen. LZMA erlaubt einerseits eine starke Kompression der Daten und ist andererseits als Java Bibliothek[24] kostenlos zu nutzen.

[23] LZMA2: Moderner Kompressionsalgorithmus [13] [10]

[24] XZ Bibliothek für Java [14]

7 Zusammenfassung und Ausblick

Rückblickend erkennt man, dass das Update-System einige der wichtigsten Anforderungen nicht erfüllt hat. Zwischen 2007 und 2010 kam es immer wieder vor, dass tausende Nutzer vom Update-System komplett abgeschnitten wurden. Der häufigste Grund dafür waren Fehler in der Aktualisierungsanwendung. Werden solche unerkannt veröffentlicht, führt dies zu einem Ausfall der kompletten Aktualisierungsanwendung auf dem Rechner der Endnutzer. Abgesehen davon kann aber auch ein unterbrochener Update-Prozess zu einem inkonsistenten Zustand führen. Ein solcher Zustand ist erreicht, wenn nur teilweise auf die neueste Version aktualisiert wurde. Ist z.B. nach einem Abbruch die HTTP-Bibliothek nicht mehr kompatibel zum Programmkern, kann dies ebenfalls zum Ausfall der Aktualisierungsanwendung führen.

Fehler wie diese wurden in der Vergangenheit meist schnell bemerkt, konnten dann aber wegen der, in Kapitel 6.1.1.3 (Seite 41), beschriebenen Problematik selten kurzfristig ausgebessert werden.

Schlussendlich wurde im Laufe der Jahre die Anzahl der Dateien in Verbindung mit langsamen oder schlecht angebundenen Internetverbindungen immer mehr zum Problem. Für einen australischen Nutzer dauert eine Aktualisierung über eine Sekunde pro Datei. Legt man das Beispiel aus Kapitel 5.2.1(Seite 23) zu Grunde, würde dieser Nutzer über acht Minuten für 500 geänderten Dateien benötigen. Die Anzahl der Plugin-Dateien stieg von 2010 bis August 2013 von 560 auf 1900 Dateien. Unser australischer Beispielnutzer würde heute für eine Aktualisierung aller Plugin-Dateien über 30 Minuten benötigen.

All dies zeigt gravierende Mängel am Grundkonzept des Update-Systems. Jede Datei einzeln zu aktualisieren stellte sich als impraktikabel heraus. Dennoch ist das System seit 2007 annähernd unverändert im Einsatz. Bei näherer Betrachtung fällt auf, dass seit Januar 2010 keine Kernkomponenten mehr aktualisiert wurden. Selbst Plugins werden inzwischen nur noch unregelmäßig, in Intervallen von einigen Wochen, aktualisiert. Die größten Probleme, die sich aus dem dateibasierenden Ansatz ergeben, verlieren weitgehend an Bedeutung wenn man auf Kern-Aktualisierungen verzichtet. Plugin-Fehler können sich nicht auf das Update-System ausbreiten. „Sackgassen“ sind deswegen kaum möglich und Update-Prozessabbrüche können im schlimmsten Fall die Funktionalität einzelner Plugins beeinträchtigen. Einzig das Problem der langen Download- und Installationszeiten bleibt.

Positiv fallen allerdings die Lastverteilung sowie die redundante und flexible Auslegung der Serverstruktur auf. Über die vergangenen fünf Jahre konnte die Serverstruktur jederzeit an die aktuellen Bedürfnisse angepasst werden. Das einfache Anlegen von Up-

date-Zweigen hat sich zudem als sehr nützlich erwiesen, um unterschiedlichen Benutzergruppen eigene Versionen anzubieten. Ansonsten ist der dateibasierende Ansatz Fluch und Segen zugleich. Während er bei vielen Dateien die Prozesse langsam und träge macht, sorgt er doch dafür, dass die komplette Anwendung über Prüfsummen auf den jeweiligen Datenbestand der Daten-Server synchronisiert wird. Selbst Modifikationen durch den Benutzer, Antivirensysteme oder externe Anwendungen können in der Regel problemlos ausgebessert werden.

Auch die grundlegende Idee, über das Update-System sowohl Nutzer als auch Entwickler zu bündeln ging auf. Bis heute gibt es keinen nennenswerten alternativen Entwicklungszweig des JDownloaders.

Obwohl das erste Update-System lange Zeit stabil lief, wird seit 2012 an einem Nachfolger gearbeitet. Die Hauptmotivation für das neue System war eine Rückkehr zu den kurzen Release-Zyklen von 2009 und dem Ansatz der „kontinuierlichen Auslieferung“. Dieses neue Update-System ging im Januar 2013 in den öffentlichen Betatest, und hat seit dem über 1400 Änderungen verteilt. Ziel des neuen Systems war es die Nachteile des ersten Systems auszugleichen, und dabei die Vorteile zu erhalten. Die Erfahrungen und neuen Anforderungen, die in Kapitel 6 beschrieben wurden, liegen diesem neuen Ansatz zu Grunde. In den ersten acht Monaten Betatest hat das neue System alle Erwartungen erfüllt.

Eidesstattliche Versicherung

Name:	Rechenmacher	Vorname:	Thomas
Matrikel-Nr.:	2105101	Studiengang:	Mechatronik

Hiermit versichere ich, Thomas Rechenmacher an Eides statt, dass ich die vorliegende Projektarbeit mit dem Titel „Online-Update-Systeme am Beispiel JDownloader" selbständig und ohne fremde Hilfe verfasst und keine anderen als die angegebenen Hilfsmittel benutzt habe. Die Stellen der Arbeit, die dem Wortlaut oder dem Sinne nach anderen Werken entnommen wurden, sind in jedem Fall unter Angabe der Quelle kenntlich gemacht. Die Arbeit ist noch nicht veröffentlicht oder in anderer Form als Prüfungsleistung vorgelegt worden.

Auszug aus dem Strafgesetzbuch (StGB)

§ 156 StGB Falsche Versicherung an Eides Statt

Wer von einer zur Abnahme einer Versicherung an Eides Statt zuständigen Behörde eine solche Versicherung falsch abgibt oder unter Berufung auf eine solche Versicherung falsch aussagt, wird mit Freiheitsstrafe bis zu drei Jahren oder mit Geldstrafe bestraft.

_________________________ _________________________

Ort, Datum Unterschrift

Quellenverzeichnis

[1] AppWork GmbH, [Online]. Adresse: http://weMakeYourAppWork.com. [Zugriff am 21 August 2013].

[2] JDownloader Team, „Subversion Repository," 08 November 2009. [Online]. Adresse: svn://svn.jdownloader.org/jdownloader/trunk | Revision 9581. [Zugriff am 16 September 2013].

[3] Google Inc., Google Trends, „Websuche-Interesse: jdownloader. Weltweit, Jun. 2007 - Jun. 2013," [Online]. Adresse: http://www.google.de/trends/explore?q=JDownloader. [Zugriff am 21 August 2013].

[4] JDownloader Team, „outdated.dat," 05 November 2009. [Online]. Adresse: svn://svn.jdownloader.org/jdownloader/trunk/ressourcen/outdated.dat | Revision 9544. [Zugriff am 27 August 2013].

[5] JDownloader Team, „Updater.java," 03 November 2009. [Online]. Adresse: svn://svn.jdownloader.org/jdownloader/trunk/src/jd/utils/Updater.java | Revision 9490. [Zugriff am 02 September 2013].

[6] JDownloader Team, „build.xml," [Online]. Adresse: svn://svn.jdownloader.org/jdownloader/trunk/build/build.xml | Revision 9311. [Zugriff am 16 September 2013].

[7] JDownloader Team, „WebUpdate.java," 03 November 2009. [Online]. Adresse: svn://svn.jdownloader.org/jdownloader/trunk/src/jd/utils/WebUpdate.java | Revision 9490. [Zugriff am 04 August 2013].

[8] JDownloader Team, „jd.update," 26 Oktober 2009. [Online]. Adresse: svn://svn.jdownloader.org/jdownloader/trunk/src/jd/update | Revision 9308. [Zugriff am 12 August 2013].

[9] A. Tridgell und P. Mackerras, „The rsync algorithm," Juni 1996. [Online]. Adresse: https://digitalcollections.anu.edu.au/bitstream/1885/40765/3/TR-CS-96-05.pdf. [Zugriff am 01 September 2013].

[10] Amazon Web Services, Inc., „Amazon CloudFront," [Online]. Adresse: http://aws.amazon.com/de/cloudfront/. [Zugriff am 06 September 2013].

[11] Google Inc., Google Project Hosting, [Online]. Adresse: https://code.google.com/intl/de/. [Zugriff am 06 September 2013].

[12] JDownloader Team, „JDownloader Website," [Online]. Adresse: http://www.jdownloader.org. [Zugriff am 21 August 2013].

[13] Jenkins Community (Kohsuke Kawaguchi), „Jenkins Projekt Website," [Online]. Adresse: http://jenkins-ci.org/. [Zugriff am 22 August 2013].

[14] W. Davison, „rsync," [Online]. Adresse: http://rsync.samba.org/. [Zugriff am 06 September 2013].

[15] J. Fredrick, P. Julius und A. Almagro , „CruiseControl Projekt Website," [Online]. Adresse: http://cruisecontrol.sourceforge.net/. [Zugriff am 22 August 2013].

[16] B. Lindholm, „New Options in the World of File Compression," May 2009. [Online]. Adresse: http://linuxgazette.net/162/lindholm.html. [Zugriff am 21 August 2013].

[17] Network Working Group, J. Postel und J. Reynolds, „File Transfer Protocol," Oktober 1985. [Online]. Adresse: http://www.ietf.org/rfc/rfc959.txt. [Zugriff am 04 September 2013].

[18] Network Working Group , „Hypertext Transfer Protocol -- HTTP/1.1," Juni 1999. [Online]. Adresse: http://www.ietf.org/rfc/rfc2616.txt. [Zugriff am 04 September 2013].

[19] I. Pavlov, „7-Zip - LZMA SDK," [Online]. Adresse: http://www.7-zip.de/sdk.html. [Zugriff am 03 September 2013].

[20] L. Collin, „The Tukaani Project," [Online]. Adresse: http://tukaani.org/xz/java.html. [Zugriff am 06 September 2013].

[21] Network Working Group, „RFC 1321 - The MD5 Message-Digest Algorithm," April 1992. [Online]. Adresse: http://tools.ietf.org/html/rfc1321. [Zugriff am 21 August 2013].

[22] Oracle Corp., „JAR File Specification," Oracle Corp., 2011. [Online]. Adresse: http://docs.oracle.com/javase/6/docs/technotes/guides/jar/jar.html. [Zugriff am 16 September 2013].

[23] B. Collins-Sussman, C. M. Pilato und B. W. Fitzpatrick, „Version Control with Subversion," 2011. [Online]. Adresse: http://svnbook.red-bean.com/en/1.7/svn-book.pdf. [Zugriff am 16 September 2013].