HSR
HOCHSCHULE FÜR TECHNIK
RAPPERSWIL
FHO Fachhochschule Ostschweiz

Clurity

Secure Wrapper on Untrusted Cloud Storage

Bachelorarbeit

Abteilung Informatik
Hochschule für Technik Rapperswil

Frühjahrssemester 2014

Autor(en):	Matthias Hauser, Philip Kaufmann
Betreuer:	Prof. Dr. Josef M. Joller
Experte:	Matthias Lips
Gegenleser:	Prof. Stefan F. Keller

Abstract

Sicherheit ist schon lange nicht mehr nur ein Thema für das reale, sondern auch für das virtuelle Leben. Ein Grossteil der Bevölkerung speichert wichtige persönliche Informationen und Dateien bereits im Internet ab. Der unbestrittene Vorteil dabei ist, dass sie jederzeit und von überall her zugänglich sind, mit dem Nachteil, dass nicht immer ersichtlich ist, was mit den gespeicherten Dateien geschieht.

Die Software Clurity versucht diesem Problem nun entgegen zu wirken. Als Java 8 Applikation bietet sie dem Benutzer die Möglichkeit, Daten lokal auf dem Microsoft Windows Betriebssystem mittels AES-256 [1] zu verschlüsseln, um sie anschliessend auf einen odere mehrere Cloud Storage Anbieter hochzuladen. Dritten bleibt somit der Einblick in die Dateien verwehrt. Damit keine Rückschlüsse über die Ordnerstruktur oder Dateigrössen gemacht werden können, werden die Dateien in Pakete zufälliger Grösse aufgeteilt, mit einem zufällig gewählten Dateinamen versehen und in einer flachen Hierarchie auf den Hoster geladen.

Clurity ist Multiclient fähig, wodurch der Benutzer von verschiedenen Computern auf seine Dateien zugreifen kann. Ermöglicht wird dies durch ein ebenfalls verschlüsseltes File, in dem die nötigen Informationen zu den auf dem Hoster liegenden Dateien stehen. Um die Konsistenz der Dateien zu gewährleisten, werden, wie bei Datenbanken, Transaktionsgrenzen gesetzt.

In Zukunft wird die Unterstützung für weitere Betriebssysteme wie Mac OS X und Linux angeboten, zudem sollen Apps für iOS und Android entwickelt werden. Ein einfaches Recoverytool wird es dem Benutzer ermöglichen, fälschlich gelöschte Dateien wiederherzustellen, sofern die entsprechenden verschlüsselten Files vorliegen.

Management Summary

Ausgangslage

Aufgrund vermehrter Verletzung der Privatsphäre im Internet steigt das Bedürfnis nach Sicherheit. Dies betrifft sowohl die Kommunikation zwischen Benutzern, als auch den Zugriff und die Verwendung der persönlichen Daten (Ferienfotos, Steuerdokumente uvm.) durch Dritte.

Es gibt bereits eine grosse Anzahl an Software, welche dem Verlangen nach Datensicherheit in der Cloud gerecht zu werden versucht. Oftmals fehlt jedoch der Blick hinter die Kulissen dieser Anwendungen, um Aussagen über die Datenhandhabung treffen zu können. Diesem Mangel versucht Clurity gerecht zu werden.

Vorgehen / Technologien

Da die Software auf verschiedenen Betriebssystemen wie Windows, OS X oder Linux lauffähig sein soll, wurde die Programmiersprache Java in der Version 8 verwendet. Diese Arbeit beschränkt sich auf die Verwendung von Windows, die Erweiterbarkeit ist jedoch sichergestellt. Für die Gestaltung der Benutzeroberfläche kam JavaFX 8 [2] zum Einsatz, welches 2014 mit Java 8 eingeführt wurde und wie dieses verschiedene nützliche Neuerungen mit sich bringt.

Des Weiteren sind zwei Cloud Storage Anbieter, Dropbox [3] und Microsoft OneDrive [4], als Machbarkeitsnachweis (Proof of concept) eingebunden. Der Grund dieser Wahl lag darin, dass Dropbox dem Entwickler ein sogenanntes SDK (Software Development Kit) zur Verfügung stellt. SDK meint in diesem Fall eine Software, welche dem Programmierer eine Abstraktion der Schnittstelle zum Fremdsystem liefert. Im Gegensatz dazu muss die Kommunikation mit Microsoft OneDrive manuell implementiert werden. So konnte gezeigt werden, dass auf verschiedene Vorbedingungen vergleichsweise einfach reagiert werden kann.

Ergebnis

Clurity bietet als quelloffene Software eine sichere und einfach bedienbare Lösung, um Daten verschlüsselt auf verschiedenen Cloud Storage Anbietern abzulegen. Der Benutzer wählt ein Verzeichnis innerhalb des Dateisystems, welches von Clurity überwacht wird. Änderungen werden registriert und entsprechend verarbeitet. Wird das Programm auf zwei Computern installiert, sorgt Clurity dafür, dass die Dateien in Synchronisation bleiben.

Da kein Server zur Kommunikation zwischen mehreren Computern zur Verfügung steht, verarbeitet ein komplexer Algorithmus den Abgleich der Daten zwischen den Clients und den Cloud Speichern.

Ausblick

Während der Entwicklung wurde stets Wert darauf gelegt, dass weitere Anbieter ohne grosse Mühe eingebunden werden können. Ebenso sollte die Portierung auf weitere Betriebssysteme, allenfalls mit gewissen Einschränkungen, möglich sein. Der Einsatz standardisierter Verschlüsselungsverfahren unterstützt dies zusätzlich.

Aufgabenstellung

Betreuer

Betreuer HSR:

- Prof. Dr. Josef M. Joller

Ausgangslage, Problembeschreibung

Die zentrale Speicherung von Daten und deren Verwaltung auf zentralen Servern (‘Clouds‘) wird immer populärer, beispielsweise für Datensicherung, Datenaustausch und vieles mehr.

Gleichzeitig stellt sich die Frage, insbesondere unter Berücksichtigung der Sammelwut der Nachrichtendienste, ob und wie diese Daten gesichert sind.

Wichtige Aspekte eines zentralen (‘Cloud‘) Datenspeichers und Kollaborations-Werkzeugs sind:

- Privatsphäre: Jeder Teilnehmer muss jederzeit wissen (oder abfragen können), was auf der Wolke vor sich geht, welche Daten dort gespeichert sind und wer diese Daten in irgendeiner Form verarbeitet (weiterleitet, sichert, verteilt, bearbeitet, löscht).

- Sicherheit: Jeder Teilnehmer muss Gewissheit haben, dass keine unerlaubten und unerwünschten Besucher auf seinem ‘Cloud‘- Speicher aktiv sind.

Heutige Anbieter geben dem Benutzer in der Regel kaum Sicherheit und gewähren auch keinen einfachen Zugriff auf Log-Daten.

Es ist auch nicht absehbar, dass Firmen wie Dropbox, Google oder Microsoft in naher Zukunft grosse Schritte in eine der zwei Richtungen (Privatsphäre, Sicherheit) unternehmen werden. Es ist dem Client überlassen, sich zu schützen und zu beurteilen, ob und wie er die Daten auf der Cloud schützen will.

Technologische Rahmenbedingungen

Da Cloud Clients auf unterschiedlichen Plattformen existieren, muss gewährleistet sein, dass die Lösung Multi-Plattform fähig ist.

Ziele der Arbeit

Hauptziel dieser Arbeit ist es zu untersuchen, inwiefern Daten verschlüsselt auf den Cloud-Speicher geladen und dort verschlüsselt abgespeichert werden können. Dabei wird Wert darauf gelegt, dass die Vorgänge 'Verschlüsselung'und 'Auswahl'der zu übermittelnden Daten für den Client einfach möglich sind.

Die Grobziele lassen sich wie folgt zusammenfassen:

- Software mit clientseitiger Verschlüsselung, Ablegen der Daten auf 1..n Hoster
- Prototyp mit Unterstützung von 2 Anbietern, davon einer mit REST, einer mit angebotenem SDK
- Implementierung für bestehende Cloud Storage Anbieter, keine Eigenentwicklung der Serverseite
- Multiplattform-Möglichkeit anbieten, aber Implementierung auf Windows
- Bereitstellen von Hooks für Weiterentwicklung
- Minimales GUI zur Unterstützung des Benutzers
- Kein Sharing, aber Mehrfachinstallation mit ein und demselben Account möglich

Wesentlich ist der Fokus auf:

- Reine Client-Seite

  Konsequenz: Anbindung an vorgegebene APIs und allenfalls Frameworks.

- Multi-Cloud

  Da die Cloud Anbieter Szene im Fluss ist, ist unklar, welche Anbieter längerfristig überleben. Konsequenz: Das System muss für mehrere Cloud-Lösungen (eventuell sogar gleichzeitig: Speicherung der verschlüsselten Daten in mehr als einer Wolke) ausgelegt und einsetzbar und erweiterbar sein

- Benutzerfreundlich

  Konsequenz: einfaches GUI, Auswahlmöglichkeit für unterschiedliche Verschlüsselungsmöglichkeiten (und Erweiterbarkeit durch Entwickler); aber auch Möglichkeit der Verwendung von voreingestellten sinnvollen Auswahlen (Standardwerte).

Ein mögliches Zusatzziel sind Recovery und Reconfigure Dienste, die es erlauben, Clients im Falle eines Ausfalls / eines Verlusts wiederherzustellen.

Zur Durchführung

Die erfolgreiche Bearbeitung dieser Aufgabe setzt Grundkenntnisse in der Programmierung verteilter Systeme voraus (praktische Beispiele, nicht bloss Literaturkenntnisse) sowie die Bereitschaft, sich in neue und neuste Ergebnisse einzuarbeiten (Desktop Anwendungen, speziell mit Windows bzw. plattform-neutral).

Mit dem Betreuer finden in der Regel Besprechungen statt. Zusätzliche Besprechungen sind nach Bedarf durch die Studierenden zu veranlassen. Das Sitzungsintervall ist flexibel: falls wenig oder keine Probleme auftreten, können die Sitzungsintervalle vergrössert werden; falls Probleme auftauchen, werden die Intervalle verkleinert.

Für die Durchführung der Arbeit ist ein Projektplan zu erstellen. Dabei ist auf einen kontinuierlichen und sichtbaren Arbeitsfortschritt zu achten. An Meilensteinen gemäss Projektplan sind einzelne Arbeitsresultate in vorläufigen Versionen abzugeben.

Dokumentation und Abgabe

Wegen der beabsichtigten Verwendung der Ergebnisse in weiteren Projekten wird auf Vollständigkeit sowie (sprachliche und grafische) Qualität der Dokumentation erhöhter Wert gelegt.

Die Dokumentation zur Projektplanung und Verfolgung ist gemäss den Richtlinien der Abteilung Informatik anzufertigen. Die Detailanforderungen an die Dokumentation der Recherche- und Entwicklungsergebnisse werden entsprechend des konkreten Arbeitsplans festgelegt.

Die Dokumentation ist vollständig auf CD abzugeben oder per Zugriff auf eine Ablage (SVN, GIT). Neben der Dokumentation sind abzugeben:

- alle zum Nachvollziehen der Arbeit notwendigen Ergebnisse und Daten (Quellcode, Buildskripte, Testcode, Testdaten usw.)
- Material für eine Abschlusspräsentation (ca. 20)

Termine

Siehe Terminplan auf der HSR Seite.

nach HSR Terminplan	Beginn der Bachelorarbeit, Ausgabe der Aufgabenstellung durch die Betreuer
1. Semesterwoche	Kick-off Meeting
2. Semesterwoche	Abgabe der Projektplanung (Entwurf), einschliesslich ggf. zu beschaffender Hardware
4. Semesterwoche	Abgabe eines vorläufigen Technologierechercheberichts und eines detaillierten Vorschlags für einen Arbeitsplan. Festlegung des Projektplans mit dem Betreuer
6. Semesterwoche	Abgabe eines Umsetzungsvorschlags für die Experimentierumgebung (Funktionsumfang, Aufwandsabschätzung). Abstimmung und Festlegung mit dem Betreuer
7. Semesterwoche	Abgabe Anforderungs- und Domainanalyse für die Experimentierumgebung
10. Semesterwoche	Review-Meeting Softwaredesign für die Experimentierumgebung
13. Semesterwoche	Vorstellung der Implementierung (Arbeitsstand)
nach HSR Terminplan	Abgabe der Kurzbeschreibung zur Kontrolle an den Betreuer
nach HSR Terminplan	Abgabe des Berichtes an den Betreuer (bis 12:00 Uhr)

Literaturhinweise

Generelle Referenzen: Siehe Vorlesung über Verteilte Software Systeme.
Weitere Literatur steht bei Bedarf zur Verfügung.

Beurteilung

Eine erfolgreiche Bachelorarbeit erhält 12 ECTS-Punkten (1 ECTS Punkt entspricht einer Arbeitsleistung von ca. 25 bis 30 Stunden). Für die Modulbeschreibung der Bachelorarbeit siehe http://studien.hsr.ch/allModules/19419_M_BAI.html.

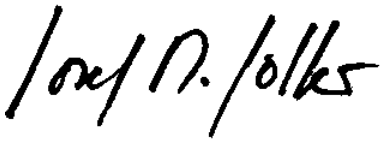
Gesichtspunkt	Gewicht
1. Organisation, Durchführung	1/5
2. Berichte (Abstract, Management Summary, techn. u. persönliche Berichte) sowie Gliederung, Darstellung, Sprache der gesamten Dokumentation	1/5
3. Inhalt *)	3/5

*) Die Unterteilung und Gewichtung von 3. Inhalt wird im Laufe dieser Arbeit mit dem Studenten vereinbart

Im Übrigen gelten die Bestimmungen der Abt. Informatik zur Durchführung von Bachelorarbeiten.

Rapperswil, den 15.09.2014

Der verantwortliche Dozent



Prof. Dr. Josef M. Joller

Inhaltsverzeichnis

1. Ausgangslage

Der Nutzen von Cloud Storage Angeboten ist unbestritten. Von überall her auf die persönlichen Daten zugreifen zu können, ermöglicht dem Benutzer eine grosse Mobilität. Gleichzeitig ist in den letzten Jahren das Misstrauen gewachsen. Die Frage nach der Datenhoheit[1] ist vielerorts spürbar und befriedigende Antworten darauf schwierig zu finden. Mit ein Grund ist der Mangel an durchgängig quelloffener Software. Oft kommt auch noch ein Server als dritter Kommunikationspartner mit ins Spiel, was einem weiteren Unsicherheitsfaktor entspricht.

Clurity, das Produkt dieser Bachelorarbeit, soll vor diesen Hintergründen dem User ein Stück Security in der Cloud zurück geben. Die Anforderungen sehen die Implementierung einer clientseitigen Verschlüsselungslösung vor, die es dem Benutzer erlaubt, seine Daten wahlweise redundant auf verschiedenen Hostern sicher zu speichern. Entsprechende Storage Anbieter wachsen derzeit wie Pilze aus dem Boden, womit eine einfache Erweiterbarkeit des Softwarecodes wünschenswert ist.

Ein grosser Vorteil von Cloudangeboten jeglicher Art ist die Plattformunabhängigkeit, welche, wenn möglich, durch Clientsoftware nicht eingeschränkt werden sollte. Daraus ergibt sich die Anforderung, dass eine spätere Portierung der Applikation auf weitere Betriebssysteme erfolgen kann.

Viele Cloud Storage Anbieter ermöglichen Data Sharing zwischen den Benutzern. Dieser Aspekt wird im Rahmen dieser Arbeit aufgrund der erhöhten Komplexität nicht verfolgt. Ein sicherer Datenaustausch kann jedoch mittels einer Zweitinstallation und einem gemeinsamen Account erreicht werden.

[1] Datenhoheit meint insbesondere, dass

- die Daten des Cloud-Nutzers jederzeit verfügbar sind,
- der Nutzer die Verfügungsbefugnis über die Daten hat,
- die Daten im jeweils genutzten System vertraulich behandelt werden,
- und das genutzte System integer ist, also geschützt ist vor überwachung, Missbrauch, oder Veränderung der darin gespeicherten Daten.[5]

2. Lösungskonzepte

2.1 Technologien

Software kann natürlich ganz verschieden erstellt werden. Im Hinblick auf spätere Portierungen fiel die Wahl der Programmiersprache auf Java. Mit der im März 2014 veröffentlichten Version 8 kamen zudem für die Arbeit interessante Neuerungen hinzu. Dazu gehören die Stream API, die Unterstützung von Lambdas und auch JavaFX 8 zur Gestaltung der Benutzeroberfläche.

Um die Software so transparent wie möglich zu halten, werden ausschliesslich offene Libraries verwendet. So kommen Komponenten verschiedener Anbieter, wie z.B. Google [6] oder der Apache Software Foundation [7], zum Einsatz.

2.2 Lokales Dateien Management

Der User soll so wenig Aufwand wie möglich haben, wenn er mit Clurity arbeitet. Das heisst, die Anwendung macht mit wenig Konfigurationsaufwand das, was der Benutzer erwartet.

Verschiedene Anbieter verfolgen mit ihrer Clientsoftware den Ansatz, einen Teilast des Dateisystems als logische Root zu verwenden. Änderungen innerhalb dieses Ordners (und seiner Unterordner) werden mit dem Cloud Speicher synchron gehalten. Clurity greift diese Idee auf und ermöglicht dem Benutzer gleichzeitig eine feingliedrige Konfiguration für den Fall, dass mehrere Hoster eingebunden sind.

2.3 Nutzung des Cloud Speichers

Verschleierung kann zur Sicherheit beitragen. In diesem Sinne bildet Clurity die verschlüsselten Daten auf eine flache Struktur in der Cloud ab. Ausserdem werden die Dateien in zufällig gewählte Grössen geteilt und umbenannt, um möglichst wenig Rückschlüsse auf die Originaldaten zuzulassen.

Da auf einen zentralen Server verzichtet wird, muss die Information über die Dateien mit den Daten selbst zusammen abgelegt werden. Diese Aufgabe übernimmt eine separate Datei, welche ebenfalls verschlüsselt auf der Cloud liegt.

2.4 Mögliche Alternativen

Sharing von verschlüsselten Daten ist ohne zentralen Schlüsselserver eine nichttriviale Aufgabe. Ein möglicher Ansatz ist der Einsatz einer distributed hash table (*dt.: Verteilte Hashtabelle* [8]), diesem wurde jedoch in dieser Arbeit nicht nachgegangen.

3. Umsetzung

Die folgenden Unterkapitel geben einen Überblick der entstandenen Architektur. Die Gliederung erfolgt Buttom-Up. Abbildung 3.1 gibt dazu eine Grobübersicht.

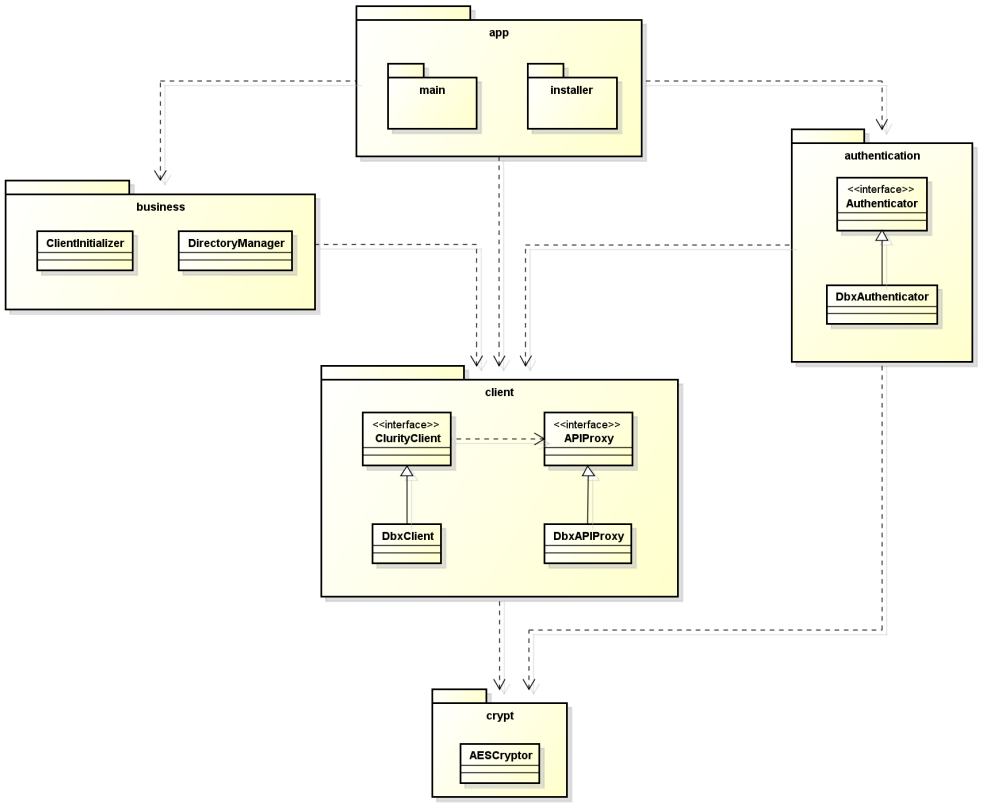


Abbildung 3.1: Übersicht Layers

3.1 Client

Abbildung 3.2 zeigt die wichtigsten Klassen des Client Packages und ihre Beziehungen untereinander. *ClurityClient* repräsentiert einen Cloud Hoster und benutzt den *APIProxy* zur Kommunikation mit letzterem. Da sich ein Grossteil der Logik für alle Clients wiederholt, wurde der *AbstractClurityClient* erstellt, von dem wiederum die konkreten Clients erben.

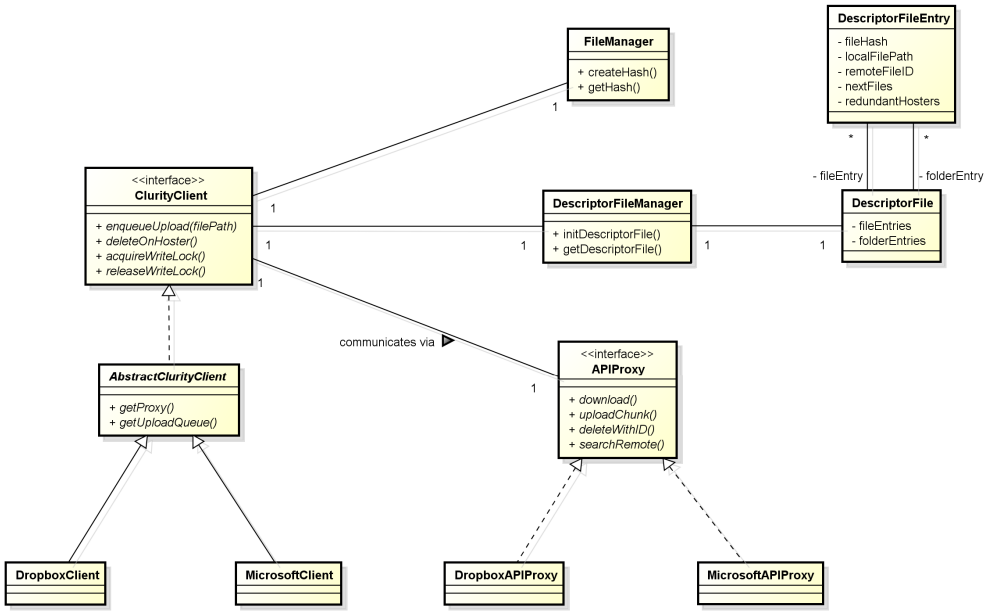


Abbildung 3.2: Abhängigkeiten im Client Package

Damit sich bei einer Mehrfachinstallation parallel laufende Clurity Instanzen nicht in die Quere kommen, wurden Transaktionsgrenzen eingeführt. Dazu stehen auf dem *ClurityClient* Interface die Methoden *acquireWriteLock* und *releaseWriteLock* zur Verfügung. Die Implementierung erstellt, bzw. löscht, dabei auf dem Cloud Speicher eine spezielle Markerdatei, dessen Existenz anzeigt, dass sich ein Programm in einer Transaktion befindet.

Die zwei implementierten APIProxys unterscheiden sich erheblich, da Dropbox, wie schon erwähnt, ein SDK zur Verfügung stellt, während im *MicrosoftAPIProxy* manuell gegen die angebotene REST-Schnittstelle [9] implementiert werden muss.

Clurity erstellt bei der Erstbenutzung einen Ordner beim Hoster, in dem alle Dateien abgelegt werden. Damit diese auf ihre entsprechenden Originaldateien abgebildet werden können, wird eine zusätzliche Datei, das *DescriptorFile* persistiert. Es enthält für jede lokale Datei einen Entry, in welchem folgende Informationen abgelegt sind:

- *fileHash*: Der vom *FileManager* ermittelte Hashwert der Datei um Änderungen zu erkennen
- *localFilePath*: Der Pfad zur Datei, relativ zum gewählten logischen Root Ordner
- *remoteFileID*: Der Identifier der remote gespeicherten Datei
- *nextFiles*: Falls die Originaldatei in Blöcke geteilt wurde, gibt diese Liste Auskunft über die IDs der zugehörigen Dateien

- *redundantHosters*: Liste der Hoster, auf denen die Datei redundant gespeichert liegt

3.2 Domänen Logik

Abbildung 3.3 zeigt die wichtigsten Klassen des Business Packages und ihre Beziehungen untereinander. Sie sind hauptsächlich verantwortlich, sich um Änderungen des Dateisystems zu kümmern und davon abgeleitete Massnahmen auszulösen. Die ganze Synchronisationslogik ist zum verbesserten Verständnis in Abbildung 3.4 stark abstrahiert dargestellt.

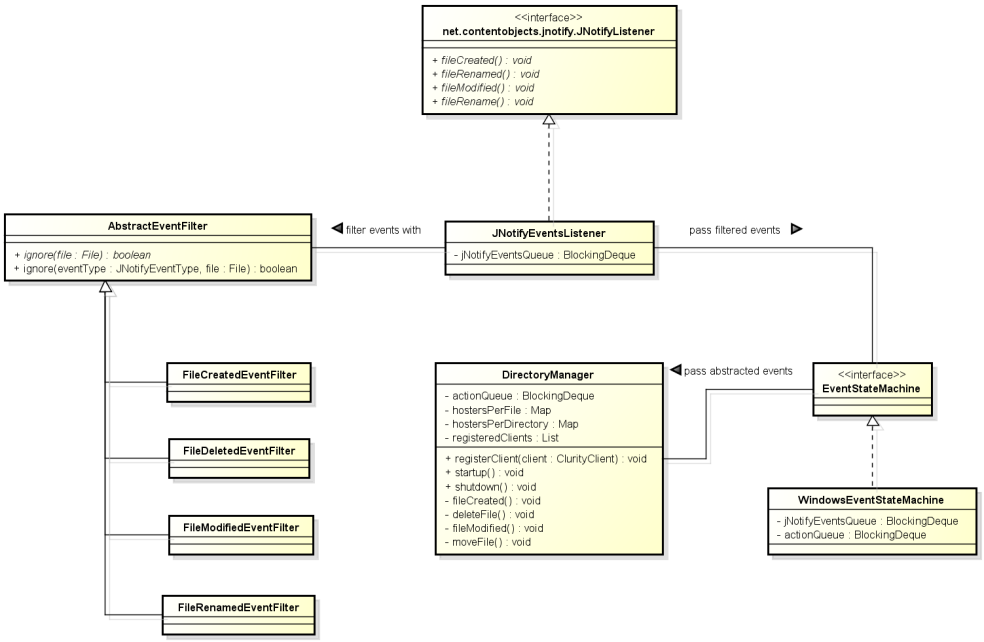


Abbildung 3.3: Abhängigkeiten im Business Package

JNotify [10] ist eine Java Library, die es erlaubt, auf Filesystemevents zu reagieren. Die Implementierung in Clurity filtert die empfangenen Events zuerst und reicht sie dann über eine Queue, die *jNotifyEventsQueue*, entkoppelt an die *EventStateMachine* weiter. Aus den empfangenen Events werden hier aussagekräftige Ereignistypen abstrahiert. Diese Abstraktionsschicht war wichtig und vor allem sehr nützlich, da je nach Operation auf dem Filesystem ein Vielzahl an Low-Level Events ausgelöst wird. Als Beispiel löst die Verschiebeoperation der Datei *c.txt* von *a/b/c.txt* nach *x/y/c.txt* auf Windows folgende Events aus:

- file deleted: a/b/c.txt
- file created: x/y/c.txt
- file modified: x/y
- file modified: x/y/c.txt
- file modified: a/b

Die *EventStateMachine* generiert daraus den vereinfachten Event

file moved: from a/b/c.txt to x/y/c.txt.

Eigentliches Herzstück der Applikation ist der *DirectoryManager*. Über die *actionQueue* erhält er die Events von der State Machine und verarbeitet sie. Mit dem Wissen, welche Datei auf welchen Hoster gehört, delegiert der DirectoryManager die zu erfüllenden Aufgaben an die registrierten Clients weiter.

Im vorhergehenden Beispiel kann der Benutzer beispielsweise konfiguriert haben, dass Dateien innerhalb des Ordners *a/* auf Microsoft OneDrive, solche innerhalb des Ordners *x/* jedoch auf Dropbox persistiert werden sollen. Der DirectoryManager veranlasst nun, dass die Datei auf Microsoft OneDrive gelöscht und auf Dropbox hochgeladen wird.

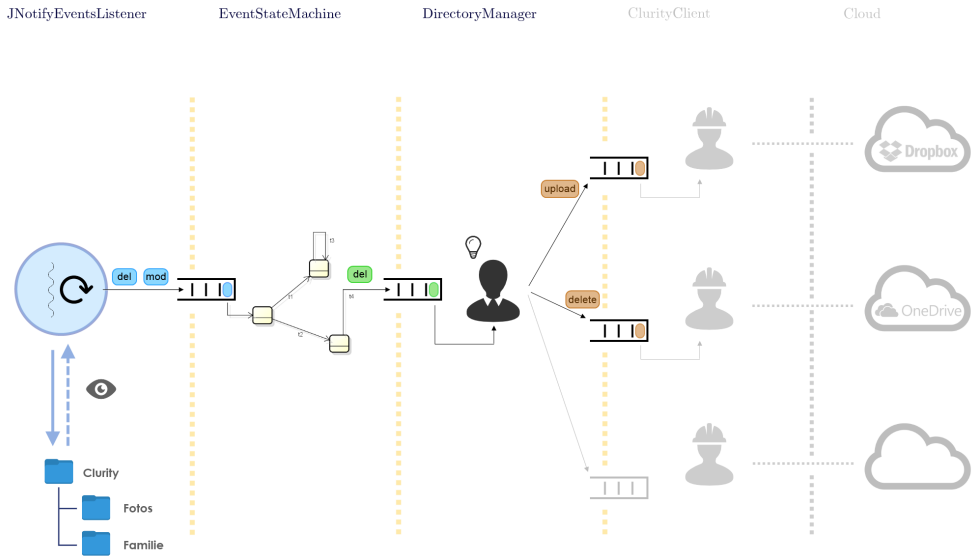


Abbildung 3.4: Synchronisationslogik abstrahiert

3.3 Authentifizierung

Clurity authentifiziert sich gegenüber den Cloud Anbietern mittels des OAuth2 [11] Verfahrens. Dieses erlaubt es der Anwendung, ohne Kenntnis der Zugangsdaten (Username, Passwort) des Benutzers, Zugriff auf dessen Daten zu erhalten. Während des Authentifizierungvorgangs wird ein sogenanntes Token erstellt, welches Clurity verschlüsselt im *TokenKeyStore* speichert. So kann sich die Applikation nach einem Neustart ohne erneute Benutzereingabe wieder beim Dienst anmelden.

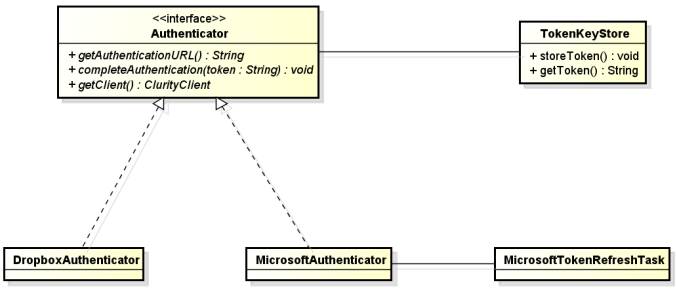


Abbildung 3.5: Abhängigkeiten Authentifizierung

Die OAuth2 Implementierung von Microsoft gibt dem Token eine Gültigkeit von lediglich einer Stunde, weshalb ein zusätzlicher Task eingeführt werden musste, der das Token aktualisiert.

3.4 Installations Wizard

Der Installations Wizard ist eine JavaFX Anwendung und führt den Benutzer durch die nötigen Schritte zur Konfiguration von Clurity. JavaFX ermöglicht es dem Programmierer, die MVC [12] Teile Model, View und Controller weitgehendst zu trennen. Abbildung 3.6 zeigt den implementierten Controller-Teil. Durch das stark zustandsabhängige Verhalten eines Installations Assistenten wurde hier die Umsetzung des State Pattern [13] gewählt.

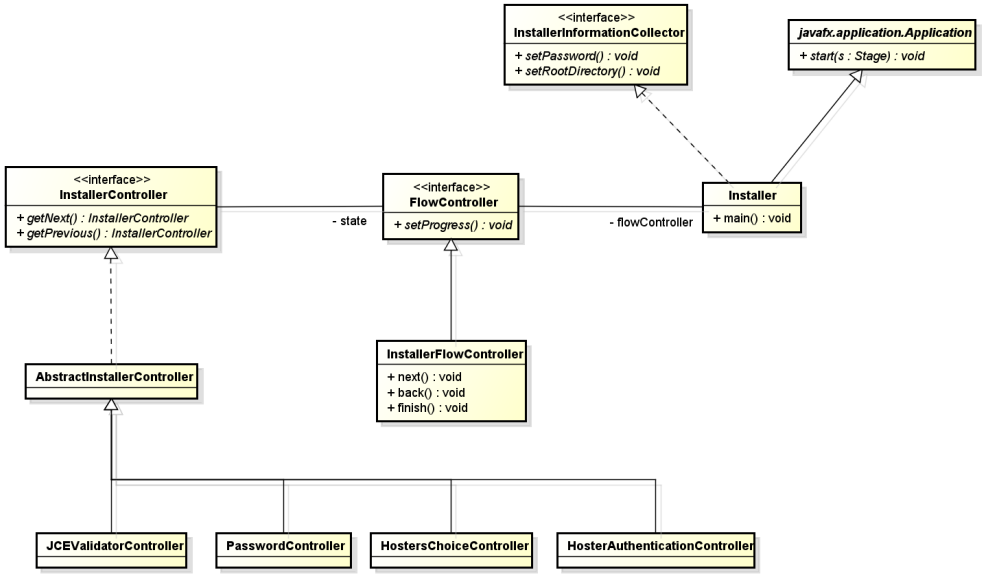


Abbildung 3.6: Installer Controller Package

3.5 User Interface

Da Clurity vor allem im Hintergrund arbeitet, ist das User Interface minimal gehalten. Styling und Dialogkomponenten wurden aus ControlsFX [14] entnommen. *HosterAuthentication*-, *MultipleChoice*- und *PasswordDialog* sind *Dialog*-Adapter, welche den Bedürfnissen der Applikation gemäss erstellt wurden.

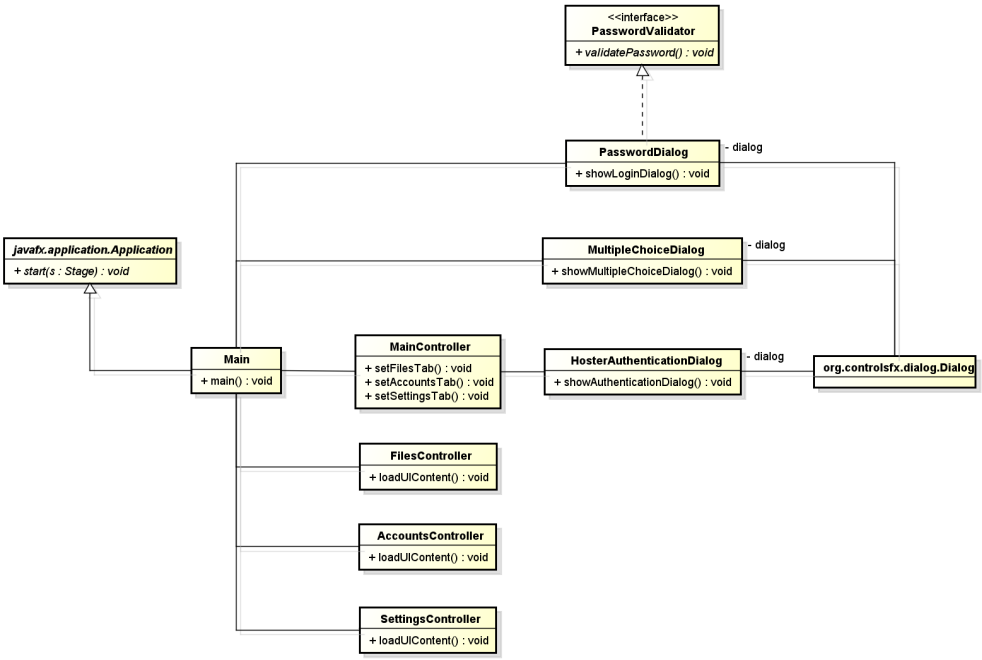


Abbildung 3.7: User Interface Package

4. Ergebnis

Die Aufgabenstellung, welche dieser Bachelorarbeit zugrunde liegt, wurde mit Clurity erfüllt. Es steht eine Software bereit, welche beliebige Benutzerdaten verschlüsselt auf bestehenden Cloud Hosting Anbietern speichert. Bei der Implementierung des zweiten Clients zeigte sich die Stärke der zuvor gemachten Logikabstraktion: innert vier Stunden waren die nötigen Methoden eingebaut und die ersten Tests erfolgreich.

Die Erkennung von Ereignissen auf dem Filesystem funktioniert ebenfalls erwartungsgemäss. Bei der Erstellung von Ordnern oder Dateien direkt im logischen Root Verzeichnis wird der Benutzer nach dem gewünschten Hoster gefragt. Dies ermöglicht eine grobe Konfiguration für den Fall, dass mehrere Hoster eingebunden sind. Folgendes Beispiel soll dies verdeutlichen:

File	Hoster	Bemerkungen
/a	H_1	Benutzer bei Erstellung gefragt
/a/x.txt	H_1	Hoster automatisch, da H_1 für Parent Folder (/a) gewählt wurde
/b	H_2	Benutzer bei Erstellung gefragt
/c/d	H_1, H_2	Benutzer bei Erstellung von c gefragt
/c/d/y.txt	H_1, H_2	Hoster automatisch, da H_1 und H_2 für c gewählt wurden

Geschehen am Filesystem Änderungen, während Clurity nicht läuft, werden diese beim nächsten Startup erkannt und mit den Hostern abgeglichen. Obwohl dies eher einer Ausnahmesituation entspricht, kann sich die Applikation auch mit fehlenden DescriptorFiles, sei es lokal oder remote, in einen stabilen Zustand hochfahren.

Das Exception handling [15] wurde wie folgt implementiert:

- Allgemeine Fehler während der Applikationsausführung:

  Clurity wird beendet. Der Benutzer wird über einen Dialog informiert. Bei Bedarf kann er den Stacktrace anschauen.

- Fehlerhaftes Verhalten eines spezifischen Clients:

  Der betroffene Client wird deaktiviert. Der Benutzer wird informiert und er kann im User Interface Details zum Fehler einsehen. Das User Interface erlaubt ihm auch, einen Restart des Clients zu versuchen.

5. Ausblick

Gewisse Aspekte konnten im Rahmen dieser Arbeit nicht berücksichtigt werden. Folgende Liste gibt einen Überblick der noch offenen Punkte:

- Mobile Applikation

  Eine mobile Anwendung sollte möglichst wenig Datenverkehr verursachen. Die Anforderung wäre hier also, eine Software, basierend auf den Kernkonzepten von Clurity, zu schreiben, welche eine Übersicht der Daten gewährt und dem User die Möglichkeit bietet, einzelne Dateien herunterzuladen.

- Portierung auf Linux, OS X

  JNotify [10] bietet schon Unterstützung für diese zwei Plattformen. Für jedes Betriebssystem müsste eine eigene Implementierung der *EventStateMachine* erstellt werden, da sich die Reihenfolge der ausgelösten Events erheblich voneinander unterscheidet.

- Recovery Tool

  Die meisten Cloud Storage Anbieter erlauben die Wiederherstellung gelöschter Dateien. Dies könnte man sich zunutze machen und basierend auf diesen Daten gewisse Recoveryfunktionalitäten anbieten.

- Internationalisierung

  Die Benutzerführung in Clurity geschieht bisher auf englisch. Java, wie auch JavaFX in Verbindung mit FXML [16], erlaubt Internationalisierung mittels Resource bundles. Davon wird bisher jedoch kein Gebrauch gemacht.

- Modularisierung

  Im Deployment-Prozess muss momentan die gesamte Applikation kompiliert werden. Der Installer hat somit die gleiche Grösse wie die Hauptapplikation, könnte aber verkleinert werden, da nicht alles gebraucht wird. Maven [17] erlaubt die Erstellung einzelner Module, welche dann je nach Anwendungszweck zusammengebaut werden können.

- Performance

  Einzelne Ansätze zur Geschwindigkeitsoptimierung wurden während der Arbeit schon realisiert. Ein weiterer, möglicher Punkt wäre das parallele Verschlüsseln der gesplitteten Dateien. Erste Tests führten jedoch Probleme bei der parallelen Benutzung der Java Cipher Klasse zu Tage.

  Wünschenswert könnte auch eine gezielte Einstellmöglichkeit der Up- und Downloadgeschwindigkeiten sein, wie dies bereits bei verschiedenen Clientanwendungen verfügbar ist.

- Open Source

  Es ist geplant, Clurity - sowohl die Anwendung, als auch der Source Code - einer breiten Öffentlichkeit zugänglich zu machen.

Abbildungsverzeichnis

Literaturverzeichnis

[1] http://de.wikipedia.org/wiki/Advanced_Encryption_Standard, abgerufen am 9.6.2014

[2] http://docs.oracle.com/javase/8/javase-clienttechnologies.htm, abgerufen am 9.6.2014

[3] http://www.dropbox.com, abgerufen am 9.6.2014

[4] http://onedrive.live.com, abgerufen am 9.6.2014

[5] http://www.lvstein.uni-kiel.de/t3/index.php?id=128, abgerufen am 7.6.2014

[6] https://code.google.com, abgerufen am 8.6.2014

[7] http://www.apache.org, abgerufen am 8.6.2014

[8] http://de.wikipedia.org/wiki/Verteilte_Hashtabelle, abgerufen am 9.6.2014

[9] http://de.wikipedia.org/wiki/Representational_State_Transfer, abgerufen am 9.6.2014

[10] http://jnotify.sourceforge.net, abgerufen am 9.6.2014

[11] http://oauth.net/2/, abgerufen am 8.6.2014

[12] http://de.wikipedia.org/wiki/Model_View_Controller, abgerufen am 9.6.2014

[13] http://sourcemaking.com/design_patterns/state, abgerufen am 9.6.2014

[14] http://fxexperience.com/controlsfx, abgerufen am 9.6.2014

[15] http://de.wikipedia.org/wiki/Ausnahmebehandlung, abgerufen am 10.6.2014

[16] http://docs.oracle.com/javafx/2/fxml_get_started/jfxpub-fxml_get_started.htm, abgerufen am 10.6.2014

[17] http://maven.apache.org, abgerufen am 10.6.2014

[18] Olaf Zimmermann - *ZIO-EC2013-W9MessageRoutingTansformationv101Lsg*, Folie 56, Institut for Software Rapperswil, 15.11.2013

[19] https://en.wikipedia.org/wiki/Known-plaintext_attack, abgerufen am 7.6.2014

[20] https://en.wikipedia.org/wiki/Initialization_vector, abgerufen am 7.6.2014

[21] https://en.wikipedia.org/wiki/Overhead_(computing), abgerufen am 7.6.2014

[22] https://en.wikipedia.org/wiki/Salt_(cryptography), abgerufen am 8.6.2014

[23] https://en.wikipedia.org/wiki/Observer_pattern, abgerufen am 8.6.2014

[24] http://www.pcmag.com/article2/0,2817,2368484,00.asp, abgerufen am 8.6.2014

[25] http://stan4j.com/, abgerufen am 10.6.2014

[26] https://en.wikipedia.org/wiki/Rational_Unified_Process, abgerufen am 10.6.2014

Anhang

A Erklärung

Wir erklären hiermit,

- dass wir die vorliegende Arbeit selber und ohne fremde Hilfe durchgeführt haben, ausser derjenigen, welche explizit in der Aufgabenstellung erwähnt ist oder mit dem Betreuer schriftlich vereinbart wurde,
- dass wir sämtliche verwendeten Quellen erwähnt und gemäss gängigen wissenschaftlichen Zitierregeln korrekt angegeben haben
- dass wir keine durch Copyright geschützten Materialien (z.B. Bilder) in dieser Arbeit in unerlaubter Weise genutzt haben.

Ort, Datum	Unterschrift
Rapperswil, 12.6.2014	
	Matthias Hauser
	
	Philip Kaufmann

B Architekturentscheide

Unten aufgeführt sind einige Architekturentscheide, welche während der Umsetzung der Arbeit getroffen wurden. Hierzu wird das Y-Template von der ABB Software Development Improvement Initiative [18] verwendet.

Im Kontext der Verschleierung der Daten und der Datensicherheit, konfrontiert mit der steigenden Erkennungsrate zusammenhängender Dateien und Ordnerstrukturen von intelligenter Software, haben wir uns für eine flache Hierarchie seitens des Cloud Storage Anbieter und gegen eine Spiegelung der Ordnerstruktur entschieden, um zu erreichen, dass so wenig Aussagen wie möglich über die gespeicherten Daten gemacht werden können. Wir nehmen dafür in Kauf, dass für die Wiederherstellung der Ordnerstruktur ein spezielles Verfahren entwickelt werden muss.

Im Kontext der Datenspeicherung und der Datensicherheit, konfrontiert mit der steigenden Erkennungsrate zusammenhängender Dateien und Ordnerstrukturen von intelligenter Software als auch der maximalen Dateigrösse für einzelne Cloud Storage Anbieter, haben wir uns für eine Aufteilung einer einzelnen Datei in verschieden grosse Teilstücke und gegen eine Aufteilung in gleich grosse Teilstücke entschieden, um zu erreichen, dass beliebig grosse Dateien abgespeichert werden können und dass keine Rückschlüsse auf die ursprüngliche Dateigrösse, Dateiformat oder Teilstückzugehörigkeit möglich sind. Wir nehmen dafür in Kauf, dass für die Wiederherstellung einzelner Dateien ein spezielles Verfahren entwickelt werden muss.

Im Kontext der Softwaretransparenz, konfrontiert mit dem steigenden Misstrauen von Benutzer gegenüber von Software, haben wir uns für eine rein clientspezifische Software und gegen einen Server entschieden, um zu erreichen, dass Clurity auch ohne Speicherung von Benutzerdaten auf einem Server funktioniert und der Benutzer sich dessen bewusst ist. Wir nehmen dafür in Kauf, dass das Teilen von Dateien mit anderen Nutzern nicht möglich ist.

Im Kontext der Datensicherheit, konfrontiert mit dem Umstand, dass der Zugriff auf die Dateien mit demselben Passwort stattfinden muss, haben wir uns für die erneute Eingabe des Passwortes zu Beginn des Programmstartes und gegen eine Speicherung des Passwortes in Klartext entschieden, um zu erreichen, dass das Passwort nicht Dritten in die Hände gelangt. Wir nehmen dafür in Kauf, dass der Benutzer auf den Komfort, das Passwort nicht ständig eingeben zu müssen, verzichten muss.

Im Kontext der Datenverschlüsselung und Datensicherheit, konfrontiert mit *Known-Plaintext* [19] Attacken, haben wir uns für den Einsatz eines Initialisierungsvektor [20] und gegen den Verzicht dessen entschieden, um zu erreichen, dass solche Attacken wirkungslos auf von Clurity gespeicherte Dateien sind. Wir nehmen dafür in Kauf, dass bei der Implementierung darauf geachtet werden muss, dass der Initialisierungsvektor beim Erstellen der Datei angehängt und beim Entschlüsseln oder Zusammensetzen zuerst ausgelesen werden muss.

Im Kontext der Passwortsicherheit, konfrontiert mit Wörterbuch- und Brute-Force-Angriffen, haben wir uns für den Einsatz eines 256bit langen Salt [22] und gegen den Verzicht dessen entschieden, um zu erreichen, dass Attacken auf Passwörter aufwändiger beziehungsweise unwirtschaftlicher sind. Wir nehmen dafür in Kauf, dass dies bei der Implementierung berücksichtig werden muss.

Im Kontext der positiven Kommunikation, wobei positive Kommunikation hier den Normalfall (kein Fehlerfall) meint, zwischen User Interface und Business logic, konfrontiert mit der Anforderung, höher liegende Layer über Änderungen informieren zu können, ohne eine Kopplung mit ihnen einzugehen, haben wir uns für den Einsatz von eigenen Listenern und gegen den von Observer entschieden, um zu erreichen, dass detailliertere Informationen beim Aufruf übergeben werden können, ohne eine Unterscheidung der Übergabeparameter wie beim Observer machen zu müssen. Wir nehmen dafür in Kauf, dass dafür eigene Listener implementiert werden müssen.

Im Kontext der Kommunikation von Expceptionhandling zwischen User Interface und Business logic, konfrontiert mit der Anforderung, höher liegende Layer über Änderungen informieren zu können, ohne eine Kopplung mit ihnen einzugehen, haben wir uns für den Einsatz vom Java bereitgestellten Observer [23] und gegen die Implementierung eines Listener entschieden, um zu erreichen, dass kein Overhead [21] implementiert wird, da die Funktion des Observers vollkommen ausreicht. Wir nehmen dafür in Kauf, dass der Observer nur für Fehlerbehandlungen zum Zuge kommt und für weitere Kommunikationen von der Business logic zum User Interface ein anderer Ansatz gewählt werden muss.

C Installationsanleitung

Clurity wird als ZIP-Datei ausgeliefert. In dieser ZIP-Datei befindet sich ein Ordner namens Clurity. Dieser Ordner sollte an jenen Ort entpackt werden, von wo aus Clurity in Zukunft gestartet wird. Im Clurity Ordner ist zu Beginn folgende Struktur vorzutreffen:

```
UnlimitedJCEPolicyJDK8/
     US_export_policy.jar
     local_policy.jar
     README.txt
config/
     DescriptorFiles/
     supportedClients.csv
resources/
     html/
     images/
     views/
lib/
jnotify/
ClurityInstaller.jar
Clurty.jar
```

Folgende Schritte müssen durchgeführt werden, um Clurity erfolgreich zu starten:

1. Die Dateien *US_export_policy.jar* und *local_policy.jar* des Ordners *Unlimited-JCEPolicyJDK8* müssen in den Ordner <Java_Intallations_Verzeichnis>/ lib/security kopiert werden.

2. Setzen der *PATH*-Umgebungsvariablen auf <Clurity_Intallations_Verzeichnis>/jnotify

3. Starten der ausführbaren Datei *Installer.jar*. Falls als Installationsordner ein Verzeichnis mit speziellen Berechtigungen gewählt wurde (z.B. C:\Programme\ Clurity), muss dies über die Konsole mit Administratorenrechten geschehen, da während der Installation neue Dateien erstellt werden. Der Installer prüft gleich zu Beginn, ob die Dateien korrekt kopiert wurden. Falls er einen Fehler anzeigt, muss Schritt 1 wiederholt werden und der Installer neu gestartet, da die JVM ansonsten die erforderlichen Dateien nicht neu lädt.

4. Auswahl eines Passworts für Clurity. Dies wird verwendet, um auf den verschlüsselten Keystore zuzugreifen, in welchem die Authentifizierungstoken von den verschiedenen Cloud Storage Anbietern gespeichert werden. Zudem wird das Passwort für die Ver- und Entschlüsselung der Dateien verwendet. Hierbei sollte auf gängige Sicherheitstipps geachtet werden [24]. Das Passwort muss jeweils zu Beginn des Programmstartes eingegeben werden.

5. Hier muss das Verzeichnis angegeben werden, von welchem Clurity die Dateien auf die später ausgewählten Hoster hochlädt. Darauf ist zu achten, dass das Verzeichnis leer sein muss.

6. Das nächste Fenster zeigt, welche Hoster zur Zeit von Clurity unterstützt werden. Den oder die gewünschten Anbieter auswählen und weiterfahren.

7. Clurity wird sich nun zu den gewählten Hostern verbinden und nach Benutzernamen und Passwort für den jeweiligen Account fragen. Dies geschieht meist über OAUTH2 [11]. Benutzernamen und Passwörter werden hierzu nicht lokal auf dem Rechner gespeichert. Lediglich die Authentifizierungstokens, welche von den jeweiligen Anbietern zur Verfügung gestellt werden, werden verschlüsselt im oben genannten Keystore abgelegt.

8. Clurity wurde erfolgreich installiert und konfiguriert und kann nun mittel der ausführbaren Datei *Clurity.jar* gestartet werden.

D FAQ

- Nach dem Start von Clurity wird eine Meldung angezeigt. Weshalb sollte ich am Ordner keine Änderungen vornehmen?

  Während des Startens von Clurity beginnt eine Synchronisationsphase, in welcher Änderungen gegenüber dem Cloud Storage Anbieter und dem lokalen Verzeichnis geprüft werden. In dieser Phase ist das Tray Icon unten rechts orange hinterlegt und eine Nachricht zeigt an, dass zur Zeit synchronisiert wird. In dieser Phase empfiehlt es sich, keine Änderungen an der Ordnerstruktur vorzunehmen, da ansonsten für den Benutzer unvorhergesehene Änderungen auftreten könnten. Das Tray Icon ändert seine Umrandung zu weiss, sobald der Vorgang abgeschlossen ist.

- Ist es möglich, Clurity auf mehreren Computern zu installieren?

  Bei einer Mehrfachinstallation auf mehreren Computern sollten wenn möglich die gleichen Cloud Storage Anbieter eingebunden werden. Clurity funktioniert jedoch auch mit einer unterschiedlichen Installation, es kann dann jedoch vorkommen, dass für den Benutzer unvorhergesehene Änderungen auftreten. Clurity versucht, den Benutzer über die Vorgänge so gut wie möglich zu informieren (zum Beispiel beim Löschen eines Anbieters).

- Wie gross dürfen die Dateien sein?

  Die Grenze liegt theoretisch bei der Kapazität des Accounts beim jeweiligen Hoster. Zu beachten ist ausserdem, dass die Zeiten für Ver- und Entschlüsselung und die Hashberechnung bei grossen Dateien stark zunehmen. Ausserdem kann das Hochladen grösserer Dateien (>300MB) bei Dropbox teils zu einem Ausfall des Dropbox-Clients führen. Clurity deaktiviert diesen anschliessend. Er kann über das User Interface jedoch ohne Probleme wieder aktiviert werden.

- Hilfe! Wie beende ich Clurity?

  Das Programm sollte immer über das Tray Icon heruntergefahren werden. Mittels Rechtsklick und *Exit* wird das Programm korrekt heruntergefahren. Es kann sonst vorkommen, dass ein Markerfile nicht gelöscht werden kann, was bei einem erneuten Start verhindert, dass sich Clurity mit dem Cloud Storage Anbieter synchronisiert.

- Die Synchronisationsphase dauert sehr lange und das Tray Icon bleibt orange. Was muss ich tun?

  Falls Clurity sich nicht synchronisiert, sei es zu Beginn, während das orange Icon angezeigt wird, oder während einer späteren Phase der Benutzung, kann es sein, dass ein Markerfile fälschlicherweise im Ordner des Cloud Storage Anbieters liegt. Gründe für dies sind unter anderem Mehrfachinstallationen, nicht korrektes Herunterfahren von Clurity oder aber teils Probleme mit dem Hoster. Falls sich der Benutzer sicher ist, dass seine eigene und weitere Instanzen von Clurity nicht mit dem Anbieter arbeitet (zum Beispiel Hochladen von Dateien), kann das Markerfile mit dem Namen *.writeLock.clurity* manuell auf Seiten des Cloud Storage Anbieters gelöscht werden. Falls sich das Programm nicht mehr in der Startphase, sondern im laufenden Betrieb befindet, kann das Markerfile auch über das Benutzerinterface gelöscht werden.

E Software Engineering Dokumente

Die während der Entwicklungsphase erstellten Dokumente, welche im Bericht sonst noch nicht erwähnt wurden, werden hier aufgelistet und beschrieben.

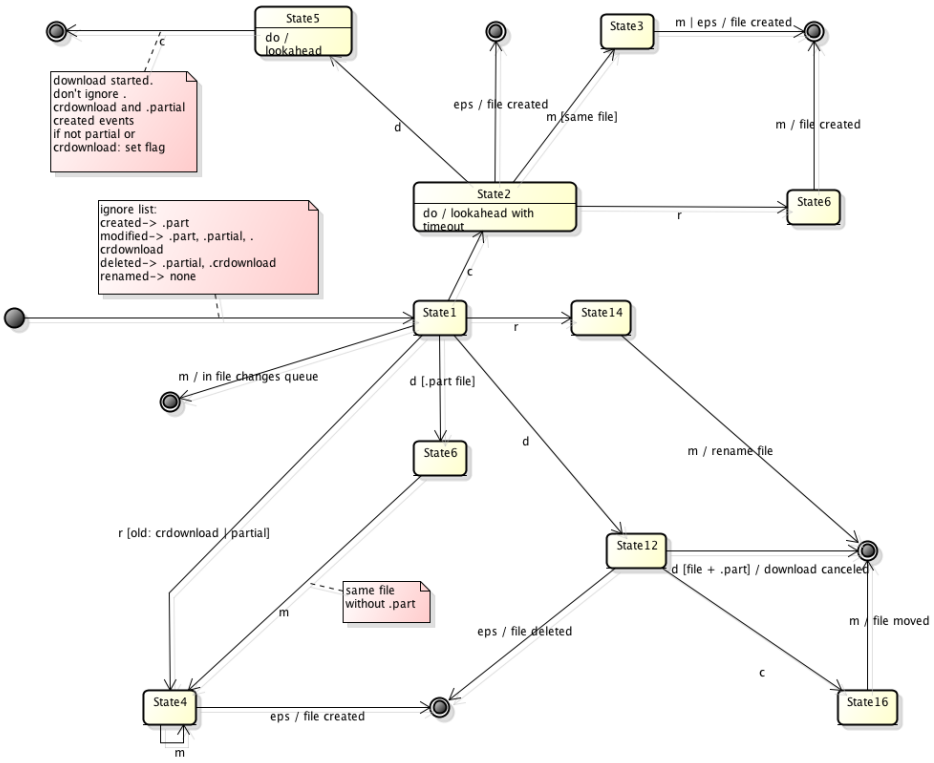


Abbildung E.1: EventStateMachine für Dateien

Abbildung E.1 zeigt die EventStateMachine, welche entwickelt wurde, um auf Dateiänderungen zu reagieren. Die Events, welche die Library JNotify [10] sendet, werden jeweils zuerst gefiltert bevor sie an die Eventstatemachine weitergereicht werden. Nach Verarbeiten eines oder mehreren Events wird das Resultat in eine Queue geschrieben, welche anschliessend weiterverarbeitet werden.

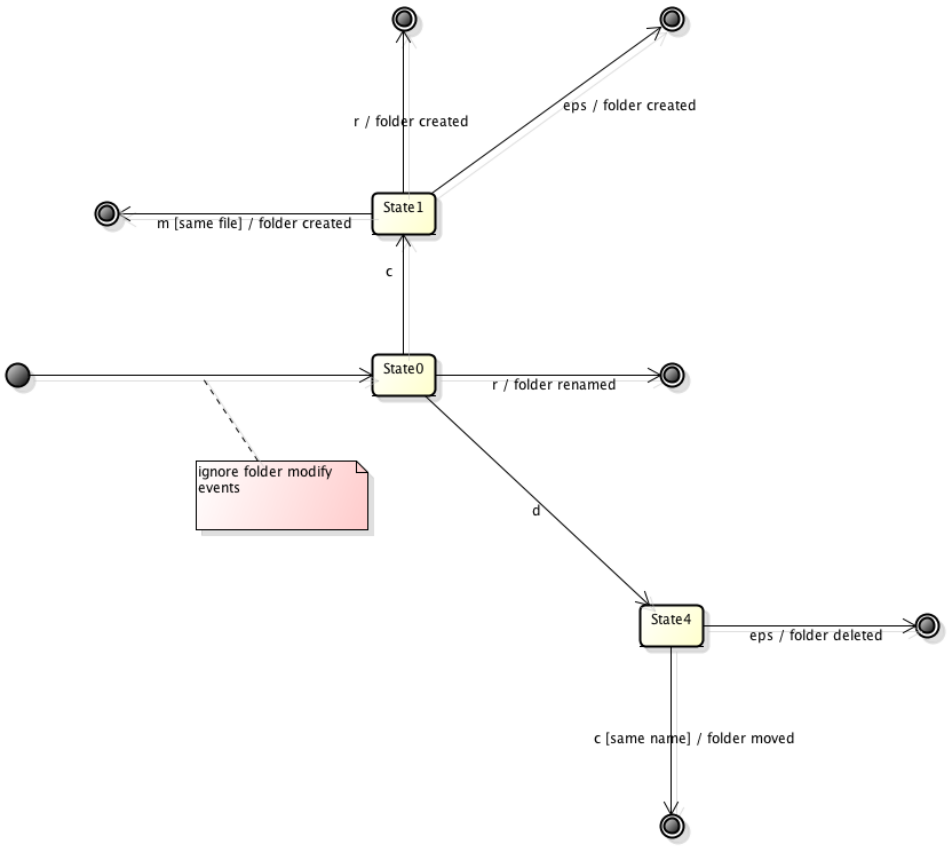


Abbildung E.2: EventStateMachine für Ordner

Abbildung E.2 basiert auf dem gleichen Prinzip wie die StateMachine aus Abbildung E.1 auf Seite 29.
Um die von der EventStateMachine produzierten Events abzuarbeiten wurden die vier nachfolgenden Aktivitätsdiagramme gemacht.

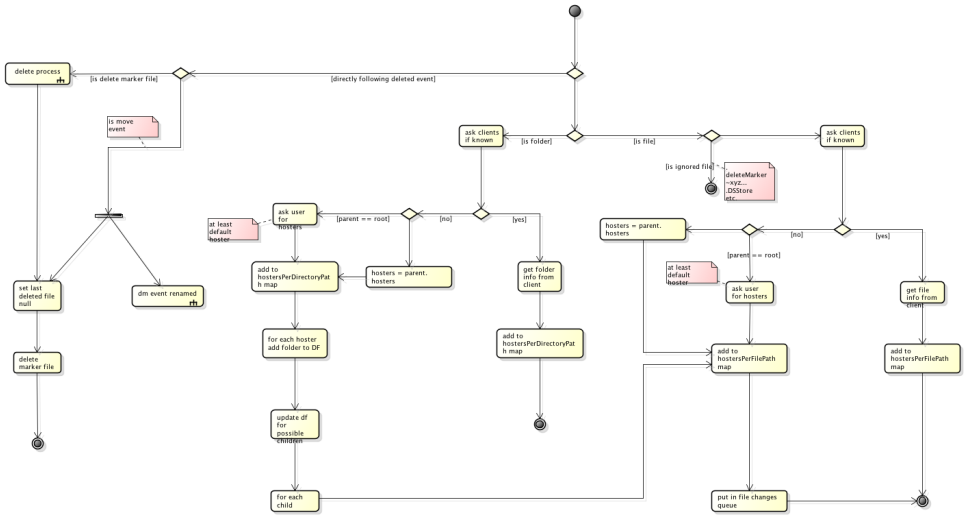


Abbildung E.3: DirectoryManager beim Erstellen eines neuen Ordners oder einer Datei

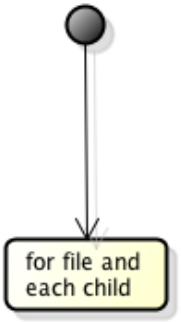


Abbildung E.4: Der zu E.3 zugehörige Löschprozess

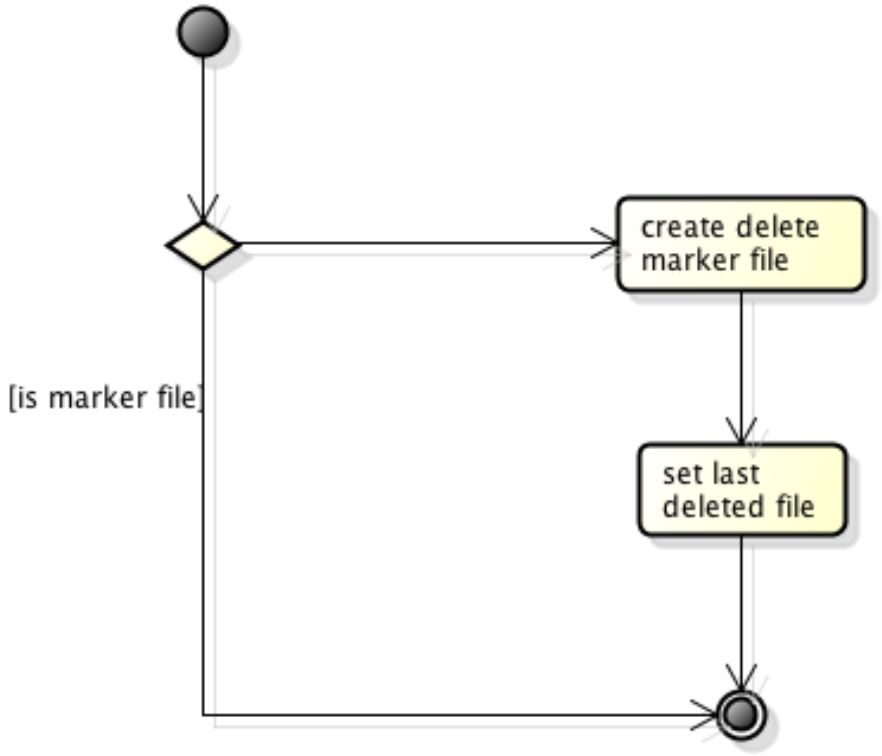


Abbildung E.5: DirectoryManager beim Löschen eines Ordners oder einer Datei

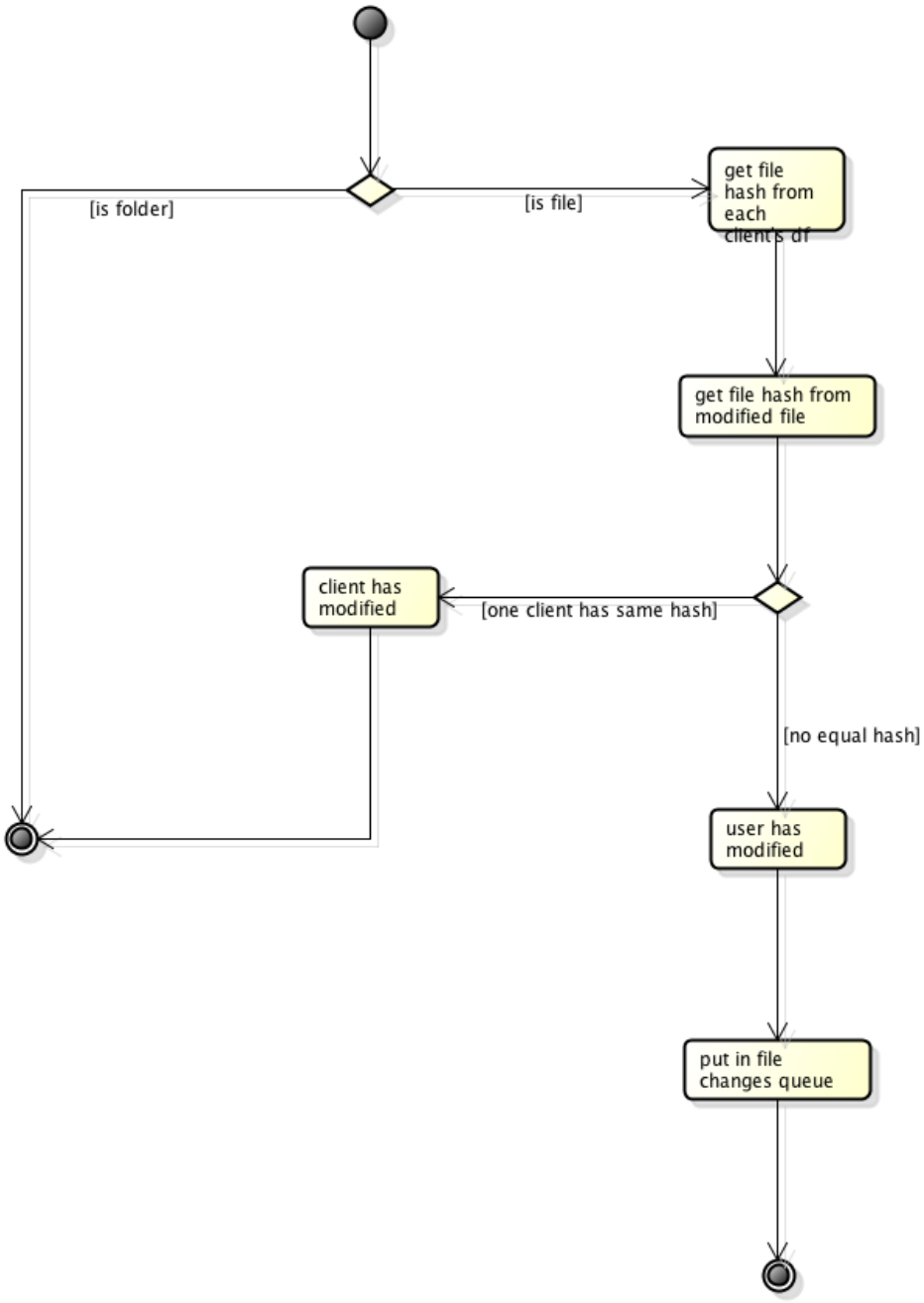


Abbildung E.6: DirectoryManager bei Änderungen einer Datei

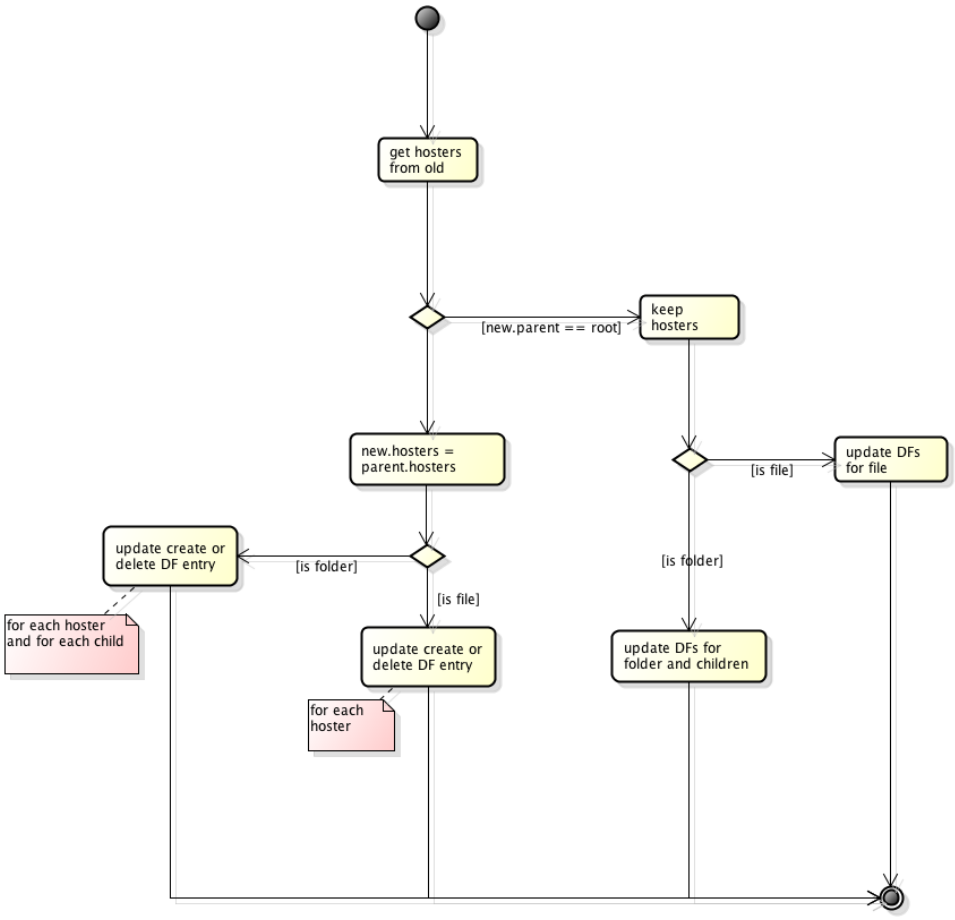


Abbildung E.7: DirectoryManager beim Verschieben oder Umbenennen eines Ordners oder einer Datei

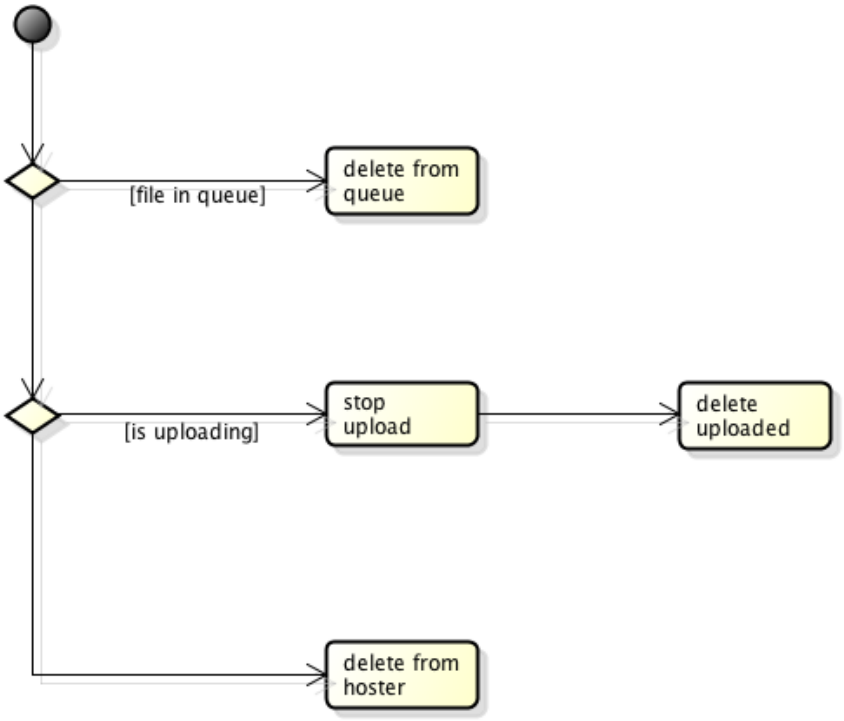


Abbildung E.8: Löschen einer Datei

Sollte der Benutzer eine Datei löschen, egal ob kürzlich erstellt oder nicht, läuft der Prozess in Abbildung E.8 ab.

Bevor der Upload einer Datei tatsächlich stattfindet, durchläuft sie einen Uploadprozess, dieser ist in Abbildung E.9 beschrieben. Diese Aufgabe übernimmt der Uploadworker.

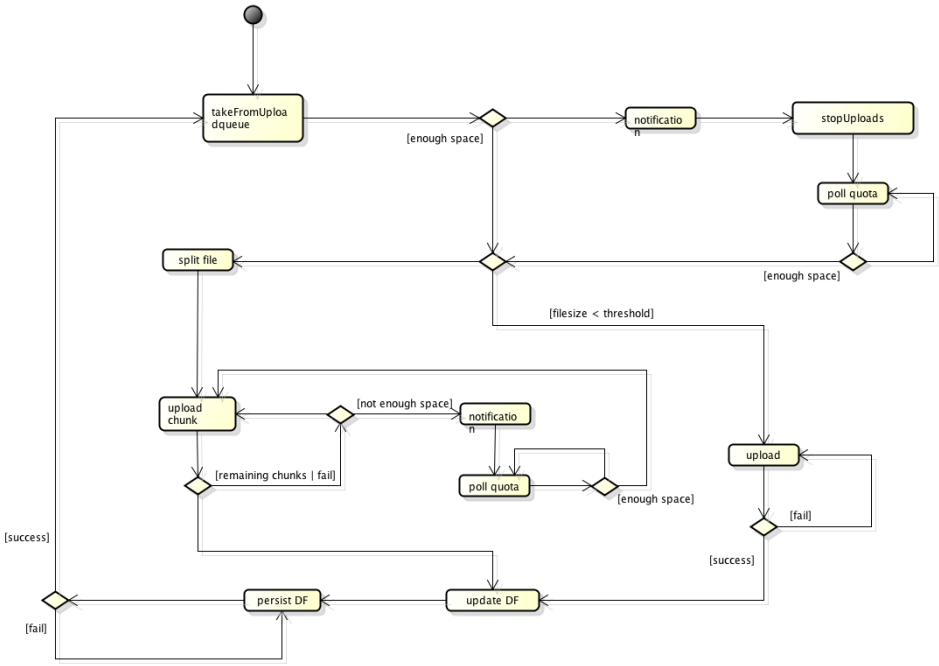


Abbildung E.9: Uploadprozess des Uploadworkers

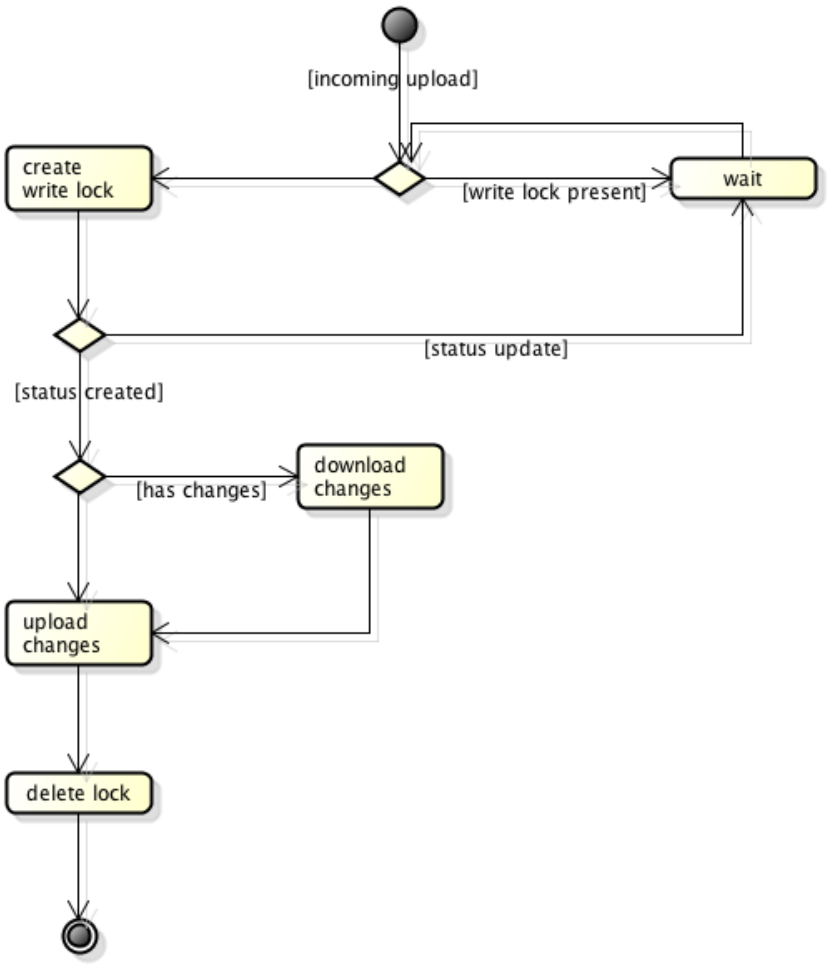


Abbildung E.10: Upload einer Datei auf einen Cloud Storage Anbieter

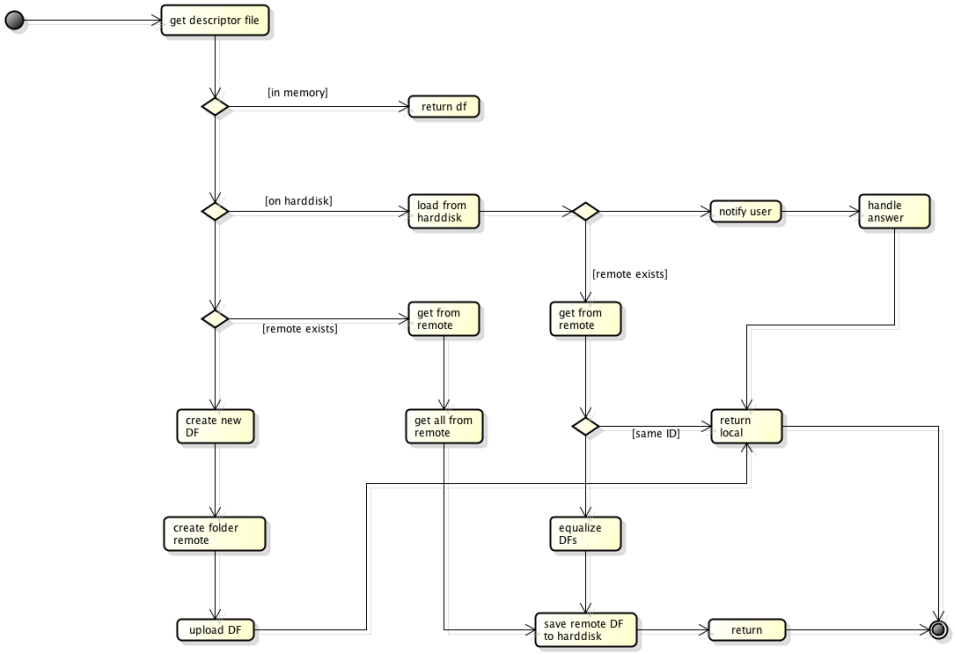


Abbildung E.11: Entscheidung für die Wahl des richtigen Descriptor Files

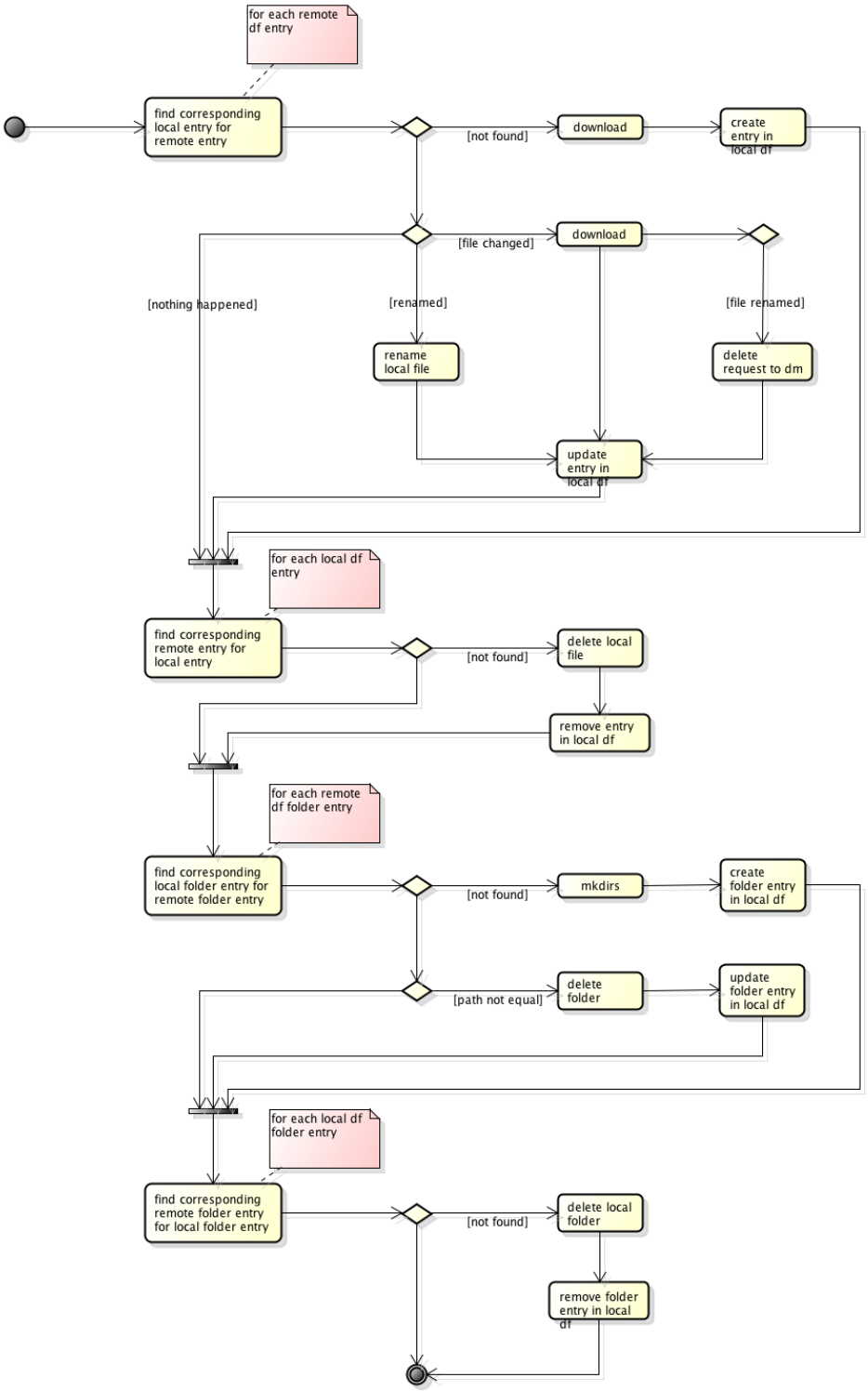


Abbildung E.12: Abgleich des lokalen Descriptor Files mit dem des Cloud Storage Hosters während des Betriebes

Beim Starten von Clurity muss jeweils überprüft werden, ob lokale Änderungen am Dateisystem stattgefunden haben. Zudem müssen mögliche Änderungen auf Seiten des Cloud Storage Hosters berücksichtig werden. Die nachfolgenden fünf Diagramme versuchen dies aufzuzeigen.

Abbildung E.13: Abgleich des lokalen Descriptor Files mit dem des Cloud Storage Hosters während des Startes

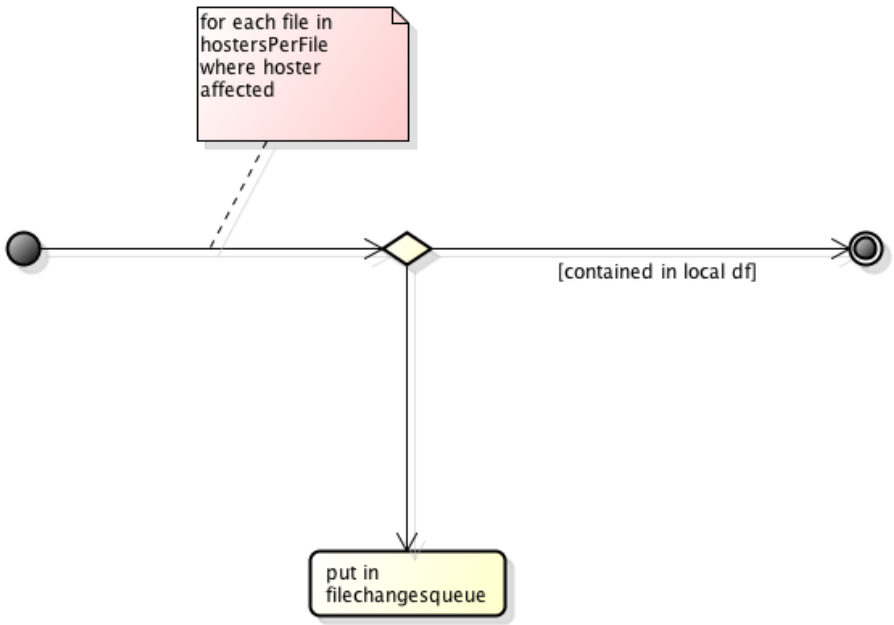


Abbildung E.14: Abgleich Dateien Lokal gegenüber Remote

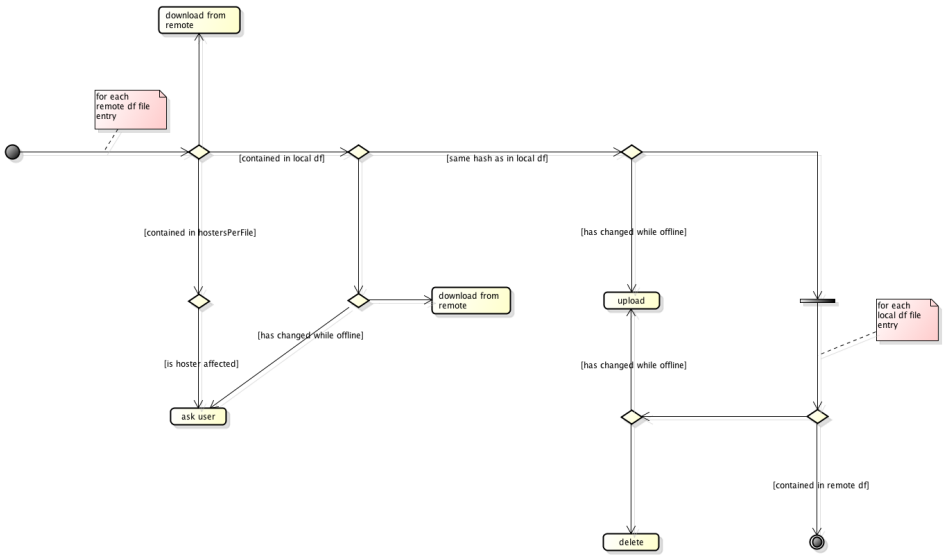


Abbildung E.15: Abgleich Dateien Remote gegenüber Lokal

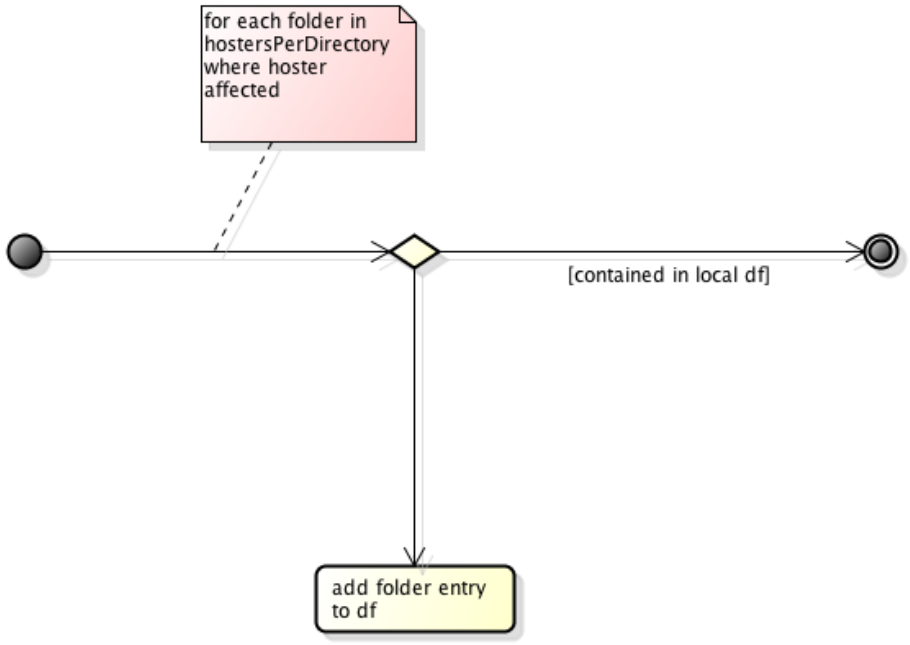


Abbildung E.16: Abgleich Ordner Lokal gegenüber Remote

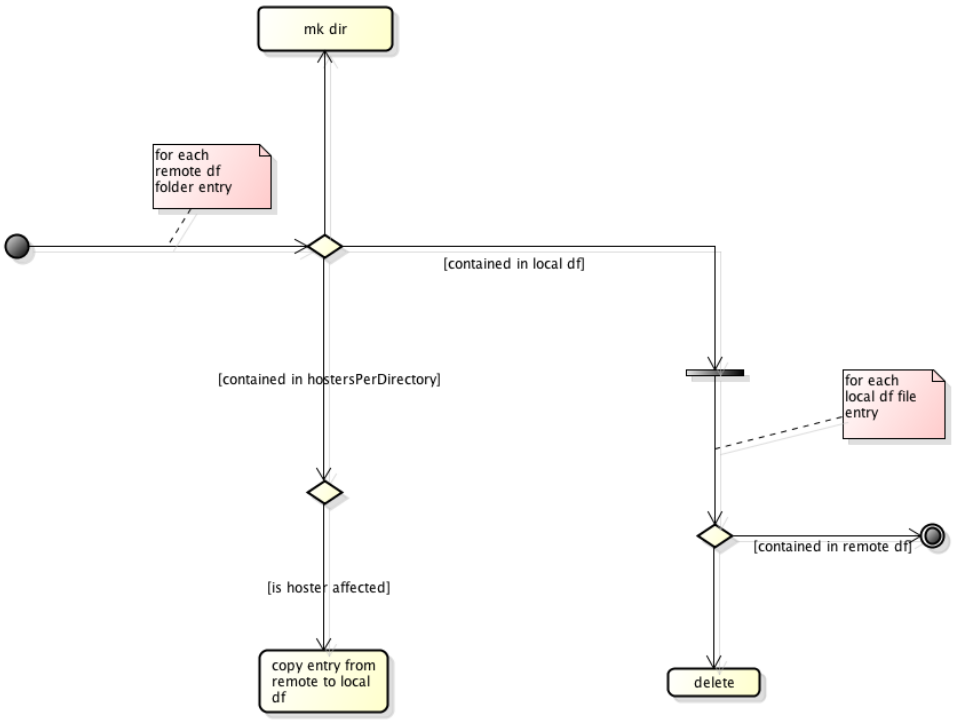


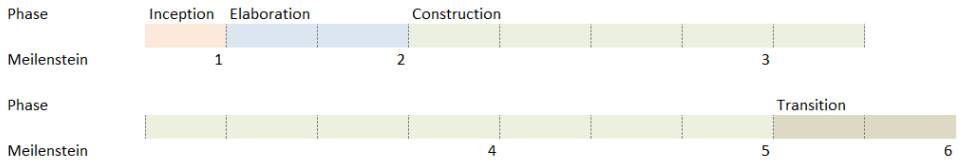
Abbildung E.17: Abgleich Ordner Lokal gegenüber Remote

F Projektmanagement

Dieser Abschnitt gibt einen Überblick über das Projektmanagement dieser Bachelorarbeit.

Projektplanung

Für die Projektplanung wurden Elemente des Rational Unified Process [26] verwendet. Folgende Abbildung zeigt die Einteilung der 17 Wochen in die Phasen *Inception*, *Elaboration*, *Construction* und *Transition*. Ein Kasten wir hierbei für eine Woche gerechnet.



Inhalte der Meilensteine:

1. Anwendungsfälle, provisorische Architektur, Entscheid Hostereinbindung
2. Protoyp für 1 Hoster (Up- und Download als Funktionalität)
3. Verschlüsselung, Erkennung Dateisystem Events
4. Zweiter Hoster, korrekter Abgleich zwischen Hostern
5. GUI, Bugfixes
6. Abgabe Bachelorarbeit

Zeitaufwand

Der für das Projekt benötigte Zeitaufwand berechnet sich wie folgt: Pro Woche wurden circa 23 Stunden gearbeitet, dies ergibt bei 15 Semesterwochen 345 Stunden pro Person. In den letzten beiden Wochen wurden jeweils 35 Stunden pro Person gearbeitet. Gesamthaft ergibt sich so ein Zeitaufwand von 415 Stunden pro Person.

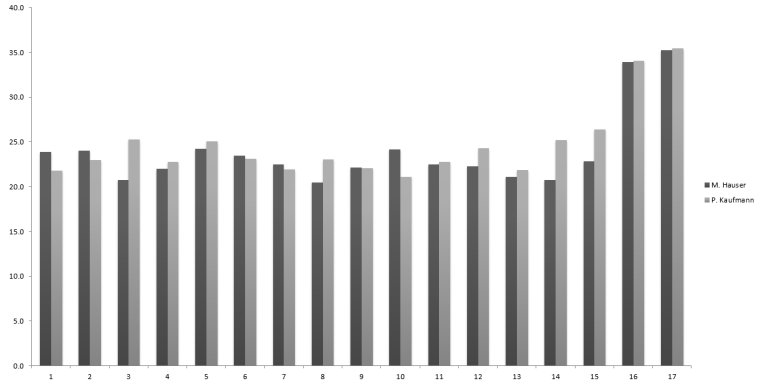
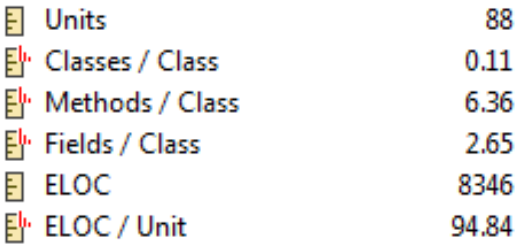


Abbildung F.18: Zeitaufwand pro Woche

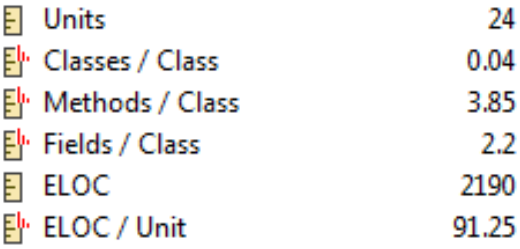
Codemetriken

Für die Erstellung der Codemetriken wurde das Eclipse Plugin STAN [25] verwendet

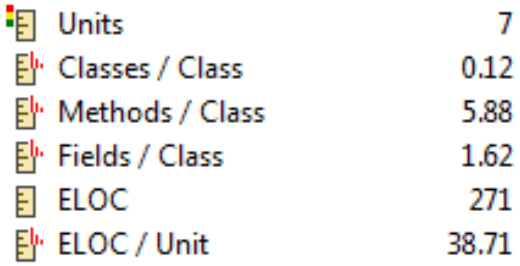
Projekt



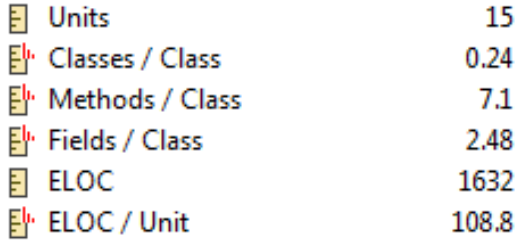
Package ch.clurity.app



Package ch.clurity.authentication



Package ch.clurity.business



Package ch.clurity.client

Units	27
Classes / Class	0.17
Methods / Class	11.92
Fields / Class	3.47
ELOC	3546
ELOC / Unit	131.33

Package ch.clurity.common

Units	3
Classes / Class	0
Methods / Class	3
Fields / Class	9.33
ELOC	102
ELOC / Unit	34

Package ch.clurity.crypt

Units	8
Classes / Class	0
Methods / Class	5.12
Fields / Class	2
ELOC	492
ELOC / Unit	61.5

Package ch.clurity.exception

Units	3
Classes / Class	0.25
Methods / Class	4.75
Fields / Class	2.5
ELOC	108
ELOC / Unit	36