BEUTH HOCHSCHULE FÜR TECHNIK BERLIN
University of Applied Sciences

Masterarbeit

Medieninformatik
Fachbereich VI – Informatik und Medien

---

# Untersuchung der Konfliktmanagementstrategien verschiedener offlinefähiger Systeme

---

Berlin, den 10. März 2018


*Autorin:*
Jacoba BRANDNER
*Matrikelnummer:*
833753

*Betreuer:*
Herr Prof. Dr. Hartmut SCHIRMACHER
*Gutachter:*
Herr Prof. Dr. John DOE

**Abstract**

In dieser Arbeit wird eine Anwendung entwickelt, die ...
Hierzu wird analysiert welche Studien, Projekte oder Anwendungen es zu diesem Thema bereits gibt. Es wird diskutiert,.... Technischen und physikalischen Grundlagen erklärt. Hierb werden Definitionen und Entwicklungswerkzeuge beschrieben und ein Überblick über mögliche Einsatzgebiete gegeben. Für das Verständnis der Umsetzung ist die Klärung der ... erforderlich. Anschließend werden die möglichen Szenarien erarbeitet, woraus sich die Anforderungen an Funktionalität und Design für die Anwendung ergeben.
Die Konzipierung und Implementierung des exemplarischen Prototyps bilden den Kern dieser Arbeit, wobei dieser Prototyp in Architektur, Funktionalität und Design erläutert und schließlich in mehreren Testreihen evaluiert wird.

**Abstract**

This work includes the development of an application ...
Therefore it is analyzed which researches, projects and applications do already exist. It is going to be discussed ... technical base ... At this point definitions and developing tools are described and an overview of potential domains is given.
For comprehension the purification of the theoretical basis of the computation is necessary. Afterwards possible scenarios are worked out what from requirements of functionality and design result.
The conception and implementation of the showcase prototype is the core of this work. This is exemplified in architecture, functionality and design and finally evaluated in several test series.

# Inhalt

# 1 Einführung

> We live in a disconnected & battery powered world, but our technology and best practices are a leftover from the always connected & steadily powered past. [off]

## 1.1 Motivation

## 1.2 Zielstellung

# 2 Grundlagen

Software, mit der ein freigegebenes Dokument zusammen mit anderen über das Internet bearbeitet werden kann, kann überaus wertvoll sein wenn man beispielsweise in einem Team arbeitet. Heutzutage gibt es viele webbasierte Software für simultanes kollaboratives Editieren *(Textdokumente, Tabellenkalkulationen, Präsentationen, Quellcode)*. Im Kapitel 3 werden Produkte und Frameworks vorgestellt, mit deren Hilfe man solche Software erstellen kann.
Lassen Sie uns einen Moment nehmen, um genauer zu definieren, was wir unter dem Begriff kollaboratives Echtzeit-Editieren verstehen. Wir möchten, dass mehrere Personen, die an verschiedenen Computern arbeiten, jederzeit Änderungen an einem auf einem Server gehosteten Dokument vornehmen können. Diese Änderungen werden sofort mit den anderen KollegInnen synchronisiert. Kein Client sollte vor einer Änderung mit dem Server oder einem anderen Client kommunizieren müssen. Insbesondere ist es nicht erforderlich, eine Sperre vom Server zu erhalten um ein Dokument zu bearbeiten und gleichzeitige Editierungen können auftreten. Nachdem alle Änderungen synchronisiert wurden, sollte jeder Client das exakt gleiche Dokument sehen.

## 2.1 Konfliktmanagementstrategien in verteilten Systemen

Es stellt sich heraus, dass die Implementierung dieser Art von Echtzeit-Zusammenarbeit alles andere als trivial ist. Im Folgenden werden die drei Strategien Operational Transformation (OT), Conflict-free replicated data type (CRDT) und Last-Write-Wins (LWW) vorgestellt.

### 2.1.1 Operational Transformation

OT als weit verbreitete Lösung stammt aus einer im Jahre 1989 veröffentlichten Forschungsarbeit [EG89]. Es wird das Problem untersucht, dass OT in einer idealen Umgebung löst und dadurch zu einem funktionierendem Algorithmus gelangt.
Ziel: Mehrere BenutzerInnen können gleichzeitig an einem Dokument arbeiten, sehen

Änderungen der anderen in Echtzeit (live), ohne dass einer Verzögerung durch die Netzwerklatenz verursacht wird. Gleichzeitig auftretende Mehrfachänderungen sollen nicht zu unterschiedlichen Dokumentenzuständen führen.

### Funktionsweise

TODO: `https://en.wikipedia.org/wiki/Operational_transformation` system architecture and basics
Jede Änderung an einem freigegebenen Dokument wird als Operation dargestellt. Operationen sind Repräsentationen von Änderungen an einem Dokument. (Beispielsweise: Füge 'Hello world!' an Position 0 in das Textdokument ein). Eine Operation zeichnet im Wesentlichen den Unterschied zwischen einer und der nachfolgenden Version eines Dokuments auf. Die Anwendung einer Operation auf das aktuelle Dokument führt zu einem neuen Dokumentstatus.
Es gibt die Operationen **das Einfügen**
Das Einfügen besteht aus dem eingefügten Text und dessen Position im Dokument (`insert('h', 0)`). Für die Position kann ein Koordinatensystem ermittelt werden (Zeilennummer: Position in Zeile oder einfacher: Dokument wie eine Folge von Zeichen behandeln, also einfach einen nullbasierten Index vergeben.)
und **das Löschen**
Löschen(5,6) = löscht 5 Zeichen, beginnend bei Position 6. Mehr benötigt man nicht, denn update = delete & insert

Um gleichzeitige Operationen zu behandeln, gibt es eine Funktion (normalerweise texttt-Transform genannt), die zwei Operationen übernimmt, die auf denselben Dokumentstatus angewendet wurden (aber auf verschiedenen Clients). Daraus wird eine neue Operation berechnet, die nach der zweiten Operation angewendet werden kann. Diese behält die erste beabsichtigte Änderung der Operation. Beispiel: Benutzer A fügt an Position 12 das Zeichen Äëin Benutzer B fügt am Anfang des Dokuments ein "Bëin. Die konkurrierenden Operationen sind daher Einfügen (12, 'A') und Einfügen (0, 'B'). Wenn wir die Operation von B einfach an Client A senden und dort anwenden würden, gäbe es ein Problem. Aber wenn die Operation von A an B gesendet, und angewandt wird nachdem Operation B angewandt wurde ist, würde das Zeichen 'A' eine Position zu weit links von der korrekten Position eingefügt werden. Dokumentstatus A und Dokumentstatus B sind nicht identisch.
Daher muss A's `insert(12, 'A')` gegen die Operation von B transformiert werden. So wird berücksichtigt, dass B ein Zeichen vor der Position 12 eingefügt hat (die die Operation `insert(13, 'A')` erzeugt.)
Diese neue Operation kann auf Dokument B nach B's Operation angewandt werden.

#### KRITIK

Während der klassische OT-Ansatz, Operationen durch ihre Versätze im Text zu definieren, einfach und natürlich zu sein scheint, werfen real verteilte Systeme ernsthafte Probleme auf. Nämlich, dass sich die Operationen mit endlicher Geschwindigkeit fortpflanzen, die Zustände der TeilnehmerInnen sind oft verschieden, so dass die resultierenden Kombinationen von Zuständen und Operationen extrem schwer vorherzusehen und zu verstehen sind. Wie Li und Li es ausdrückten: Äufgrund der Notwendigkeit, eine komplizierte Fallabdeckung in Betracht zu ziehen, sind formale Beweise sehr kompliziert und fehleranfällig, selbst für OT-Algorithmen, die nur zwei charakteristische Primitive behandeln (Einfügen und Löschen)". [LL10]
Damit OT funktioniert, muss jede einzelne Änderung an den Daten erfasst werden: Ëinen Schnappschuss des Zustands zu erhalten, ist normalerweise trivial, aber das Erfassen von Bearbeitungen ist eine ganz andere Sache. [...] Der Reichtum moderner Benutzerschnittstellen kann dies problematisch machen, besonders in einer browserbasierten Umgebung". [Fra09]

### 2.1.2 CONFLICT-FREE REPLICATED DATA TYPE

CRDTs sind Objekte, die ohne teure Synchronisation aktualisiert werden können. Sie konvergieren schließlich, wenn alle gleichzeitigen Aktualisierungen kommutativ[1] sind und wenn alle Aktualisierungen schließlich von jeder Replik ausgeführt werden [SPBZ11a]. Um diese Garantien zu geben, müssen diese Objekte bestimmte Kriterien erfüllen, welche im Folgenden beschrieben werden [SPBZ11b].

#### ZUSTANDBASIERTER ANSATZ

Wenn ein Replikat ein Update von einem Client empfängt, aktualisiert es zuerst seinen lokalen Status und dann, einige Zeit später, seinen **vollständigen Status**. So sendet jedes Replikat gelegentlich seinen vollständigen Status an ein anderes Replikat im System. Um ein Replikat, das den Status eines anderen Replikats empfängt, wendet eine **Zusammenführungsfunktion** (merge) an, um den empfangenen Status mit dem lokalen Status zusammenzuführen. Entsprechend sendet dieses Replikat gelegentlich auch seinen Status an ein anderes Replikat, sodass jedes Update schließlich alle Replikate im System erreicht.

[1] Unverändert bei Vertauschen der Operanden

### Operationsbasierter Ansatz

Bei diesem Ansatz sendet ein Replikat seinen vollständigen Status (kann groß sein) nicht an ein anderes Replikat. Stattdessen sendet es nur den **Aktualisierungsvorgang** an **alle** anderen Replikate im System und erwartet von ihnen, dass sie das Update auf sich anwenden.
Da es sich um einen Sendevorgang handelt, wenn zwei Updates $u1$ und $u2$, bei einem Replikat $i$ angewendet werden und diese Updates an zwei Replikate $r1$ und $r2$ gesendet werden, können diese Updates in unterschiedlicher Reihenfolge bei diesen replikaten ankommen. $r1$ kann sie in der Reihenfolge $u1, u2$ empfangen, während bei $r2$ die Updates in umgekehrter Reihenfolge ($u2, u1$) ankommen können. Sind die Aktualisierungen **kommutativ**, können die Repliken zussamengeführt werden, egal in welcher Reihenfolge die Updates bei ihnen ankommen - der resultierende Zustand ist derselbe. In diesem Modell wird ein Objekt, für das alle gleichzeitigen Aktualisierungen kommutativ sind, CmRDT (commutative replicated data type - kommutativ replizierter Datentyp) genannt.

**Beispiel:...**
CRDTs befassen sich mit einem interessanten und grundlegendem Problem in verteilten Systemen, haben jedoch eine wichtige Einschränkung: "Da ein CRDT konstruktionsbedingt keinen Konsens verwendet, hat der Ansatz starke Einschränkungen; Dennoch sind einige interessante und nicht-triviale CRDTs bekannt- [SPBZ11b]. Die Einschränkung ist, dass die CRDT-Adresse nur einen Teil des Problemraums betrifft, da nicht alle möglichen Aktualisierungsoperationen kommutativ sind und daher nicht alle Probleme in CRDTs umgewandelt werden können. Auf der anderen Seite können CRDTs für einige Arten von Anwendungen durchaus nützlich sein, da sie eine nette Abstraktion zur Implementierung repliziter verteilter Systeme bieten und gleichzeitig theoretische Konsistenzgarantien bieten.

### 2.1.3 Last-Write-Wins (LWW)

## 2.2 CouchDB/PouchDB

### 2.2.1 CouchDB

### 2.2.2 PouchDB

## 2.3 Offline First

1. Separate Apps from Data
2. Deliver App Code (and make it cachable) appcache & ServiceWorkers
3. Save Data Offline localStrogare / localForage, IndexedDB, other Wrapper
4. Detect Connectivity navigator.onLine (Lie-fi)
5. Sync Data - Build upon existing solutions –CouchDB/PouchDB | remoteStorage

# 3 Bestehende offlinefähige Systeme / Konzepte

## 3.1 Kollaborative Software

Kann ich benutzen um kollaborativ zu arbeiten

### 3.1.1 Google Docs

(benutzt OT)

### 3.1.2 Google Wave

### 3.1.3 Dropbox

### 3.1.4 Kollaborative Editoren

Wiki: `https://en.wikipedia.org/wiki/Collaborative_real-time_editor`

`https://atom.io/packages/covalent`
`https://atom.io/packages/firepad`
Markdown: `https://hackmd.io/`
LaTeX: `https://www.sharelatex.com/`
Online editor: `http://etherpad.org/` (OT)
Mockingbird (tool for creating wireframes): `https://gomockingbird.com/home` (OT)
marvelapp? `https://marvelapp.com/collaboration/`
–> Zahl steigend, nachdem Google die Drive Realtime API veröffentlicht hat, die auf OT basiert und es *third-party Apps* ermöglicht, dieselbe Zusammenarbeit wie Google Docs zu verwenden.

## 3.2 Offline-First Frameworks

Ich möchte aber auch eigenständig Software entwickeln die man vielleicht nicht nur zum Arbeiten nehmen kann, sondern auch um Quatsch zu machen wie Katzengifs zu teilen.

### 3.2.1 redux-offline

### 3.2.2 Realm

[rea]

### 3.2.3 hoodie

[hoo]

# 4 Konzeption

Die erarbeiteten Anforderungen an ... werden in diesem Kapitel für die Konzeption angewendet. Beginnend mit dem Aufbau der Anwendung werden in den folgenden Abschnitten die Anwendungsfälle, die Architektur und schließlich die Komponenten der Entwicklumgsumgebung aufgeführt.

## 4.1 Anwendungsaufbau

## 4.2 Architektur

## 4.3 Testfälle

Folgende Testfälle werden während der Entwicklung stetig durchgeführt. Das erfolgreiche Bestehen dieser Tests ist eine notwendige Qualitätseigenschaft der zu entwickelnden Applikation.

## 4.4 Entwicklungsumgebung

Für die Erstellung der Smartphone-Applikation wurde folgende Soft- und Hardware verwendet:

### Software

- Sublime Editor 3
- git, Version 2.7.4 zur Versionsverwaltung

### Hardware

- Tuxedo (Intel® Core™i7-6500U, 2,50GHz x 4, 7,7 GB RAM) als ersten Entwicklungsrechner (Betriebssystem: Ubuntu[1] 16.06, 64-bit-Version)

[1] Download unter `https://www.ubuntu.com/download/desktop`

- Lenovo Thinkpad X200 (Intel® Core™2 Duo, 2,40GHz, 8GB RAM) als zweiten Entwicklungsrechner (Betriebssystem: Debian 7.8, 64-bit-Version)
- Testgeräte...

# 5 Der Prototyp der Anwendung

## 5.1 Simulator Projektstruktur

## 5.2 Hauptkomponente... wie passiert was

## 5.3 Installationsanleitung

# 6 Evaluation

Um die Funktionalität des Prototyps zu untersuchen, wurden folgende Testgeräte ausgewählt.

## 6.1 Systemtest und Ergebnisse

# 7 Zusammenfassung und Ausblick

Das System ist bei Bedarf auf verschiedene Weisen erweiterbar...

# Abkürzungen

| | |
|---|---|
| CRDT | Conflict-free replicated data type |
| LWW | Last-Write-Wins |
| OT | Operational Transformation |

# Glossar

**Netzwerklatenz**

Die Wartezeit, die im Netzwerk verbraucht wird bevor eine Kommunikation beginnen kann, wird als Netzwerklatenz oder nur Latenz bezeichnet

# ABBILDUNGSVERZEICHNIS

# LITERATURVERZEICHNIS


[EG89] ELLIS, C. A. ; GIBBS, S. J.: Concurrency Control in Groupware Systems. In: *SIGMOD Rec.* 18 (1989), Juni, Nr. 2, 399–407. http://dx.doi.org/10.1145/66926.66963. – DOI 10.1145/66926.66963. – ISSN 0163–5808 2.1.1

[Fra09] FRASER, Neil: Differential Synchronization. Version: Januar 2009. https://neil.fraser.name/writing/sync/eng047-fraser.pdf. 2009. – Research Report. – 8 S. 2.1.1

[hoo] *Hoodie – The Offline First Backend*. http://hood.ie,. – Zugriff: 12.04.2018 3.2.3

[LL10] LI, Du ; LI, Rui: An Admissibility-Based Operational Transformation Framework for Collaborative Editing Systems. In: *Comput. Supported Coop. Work* 19 (2010), Februar, Nr. 1, 1–43. http://dx.doi.org/10.1007/s10606-009-9103-1. – DOI 10.1007/s10606–009–9103–1. – ISSN 0925–9724 2.1.1

[off] *Offline First*. http://offlinefirst.org/,. – Zugriff: 12.04.2018 1

[rea] *realm – The new standard in data scnchronization*. https://realm.io/,. – Zugriff: 12.04.2018 3.2.2

[SPBZ11a] SHAPIRO, Marc ; PREGUIÇA, Nuno ; BAQUERO, Carlos ; ZAWIRSKI, Marek: A comprehensive study of Convergent and Commutative Replicated Data Types / Inria – Centre Paris-Rocquencourt ; INRIA. Version: Januar 2011. https://hal.inria.fr/inria-00555588. 2011 (RR-7506). – Research Report. – 50 S. 2.1.2

[SPBZ11b] SHAPIRO, Marc ; PREGUIÇA, Nuno ; BAQUERO, Carlos ; ZAWIRSKI, Marek: Conflict-free Replicated Data Types. Version: Juli 2011. https://hal.inria.fr/inria-00609399. 2011 (RR-7687). – Research Report. – 18 S. 2.1.2, 2.1.2

# Anhang

## Eidesstattliche Erklärung

## CD-Inhalt

Auf der beigefügten CD befinden sich

- Die schriftliche Ausarbeitung dieser Masterrarbeit im PDF-Format
- Das erstellte Projekt