BEUTH HOCHSCHULE FÜR TECHNIK BERLIN
University of Applied Sciences

Masterarbeit

Medieninformatik
Fachbereich VI – Informatik und Medien

Untersuchung der Konfliktmanagementstrategien verschiedener offlinefähiger Systeme

---

Berlin, den 14. April 2018

Autorin:
Jacoba BRANDNER
Matrikelnummer:
833753

Betreuer:
Herr Prof. Dr. Hartmut SCHIRMACHER
Gutachterin:
Frau Prof. Dr. Petra SAUER

Abstract

In dieser Arbeit wird ...

Abstract

This work includes ...

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]

Heutzutage besitzen mehr als fünf Milliarden Menschen ein Mobiltelefon und drei Milliarden haben Zugang zum Internet [Ban16].

langsame Verbindungen, Unterbrechungen. Auch bei 3G und 4G ist die Latenz schrecklich (Lie-Fi?) -> Offline-First Apps können eine bessere User experience bieten.

1.1 Motivation

Ich möchte eine offlinefähige (mobile?) Anwendung entwickeln und stelle mir folgende Fragen.
Welche Software/ Framework benutze ich dazu?
Auf was muss ich bei der Auswahl achten?
Was erwarte ich von einer offline fähigen App?
(funktioniert und kein Datenverlust) –> Synchronisation und Konfliktmanagement

1.2 Zielstellung

Wichtig: Offline nutzbar ohne Datenverlust.
Untersuchung des Verhaltens bei Konflikten (verursacht durch paralleles Arbeiten ohne Internetverbindung).
Wie leicht/schwer ist es zu implementieren? konkret werden. Definition offlinefähig

2 Grundlagen

Was bedeutet offlinefähig?

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 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

2.1.1 Anforderungen an Offline-First / PWA?

1. Lese- und Schreibfähigkeit
2. Sync im Hintergrund

3. Cache sollte in der Lage sein App-Aktualisierungen zu “überstehen“

2.1.2 Progressive Web Apps?

ServiceWorker?

localForage || AsyncStorage?

IndexedDB?

2.2 Konflikte

Verteilte Systeme: Das ist ein mächtiger Begriff für viele Ideen und Konzepten, aber es läuft in der Regel darauf hinaus: Da sind zwei oder mehr Computer, die durch ein Netzwerk verbunden sind und es wird versucht, dass einige der Daten auf beiden Computern gleich aussehen. ==> Ein System das zuverlässig über ein Netzwerk funktioniert.
Zwei Geräte, ein Server, über Netzwerk verbunden.

Spezielle Eigenschaft von Netzwerken: Verbindung kann jederzeit abbrechen: Acht Irrtümer der verteilten Datenverarbeitung:

1. Das Netzwerk ist zuverlässig
Der Strom kann ausfallen oder Glasfaserkabel können kaputt sein — Das Netzwerk ist nicht zuverlässig.

2. Die Latenz ist gleich null
Glasfaserkabel werden durch Mikrowellen (oder andere Technologien) ersetzt um Millisekunden an Zeit zu sparen. Das würde nicht passieren, wäre die Latenz bei null. Es dauert nun mal eine gewisse Zeit(ms) wenn ein Signal eine (geografisch)weite Strecke zurücklegen muss — Die Latenz ist nicht gleich null.

3. Die Bandbreite ist unendlich
Daten können nicht schneller fließen als die Komponenten die sie verarbeiten (Middleware, Datenbank …) — Die Bandbreite ist nicht unendlich.

4. Das Netzwerk ist sicher
Der Heartbeat-bug[1], der im Jahr 2014 behoben wurde und die Sicherheitslücke im ICE-Wireless Local Area Network (WLAN) im Jahr 2016[2] sind nur zwei Beispiele die zeigen, dass das Netzwerk nicht sicher ist.

[1] `http://heartbleed.com/` - Zugriff: 07.04.2018
[2] `https://netzpolitik.org/2016/datenschutz-im-zug-deutsche-bahn-will-sicherheitsluecke-in-neuem-ice-wlan-schliessen/` - Zugriff: 07.04.2018

5. Die Netzwerkstruktur wird sich nicht ändern
Eine Datenbank kann beispielsweise über mehrere Server verteilt sein, die (teilweise) voneinander abhängig sind. Ein Server mit Abhängigkeiten kann ausfallen, es kann eine Aktualisierung für einen anderen Server geben — die Struktur ändert sich.

5. Die Netzwerkstruktur wird sich nicht ändern
Eine Datenbank kann beispielsweise über mehrere Server verteilt sein, die (teilweise) voneinander abhängig sind. Ein Server mit Abhängigkeiten kann ausfallen, es kann eine Aktualisierung für einen anderen Server geben — die Struktur ändert sich.

6. Es gibt eineN AdministratorIn
Es kann beliebig viele AdministratorInnen geben.

7. Die Datentransportkosten sind gleich null
Netflix bezahlte anfang 2014 diversen InternetanbieterInnen dafür, dass Netflix KundInnen bevorzugten Internetzugang haben.

8. Das Netzwerk ist homogen
Es gibt verschiedene Arten von Netzwer: 3G, 4G, LTE, WiFi. Wird beeinflusst durch Hardware (Smartphone, Tablet, PC, Laptop, Router …) [fal]

2.2.1 CAP Theorem

2.3 Replikation 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 plus CouchDBs Peplikationsmodell.

2.3.1 Last-Write-Wins (LWW) || Blockieren?

NoSQL bietet dies, indem es einen Zeitstempel von irgendeiner Art beibehält, der ihnen hilft, zu entscheiden, welcher Schreibvorgang zuletzt kam. Datenbanken wie DynamoDB[3] oder Cassandra [4] verwenden LWW, um Schreibvorgänge zu verarbeiten. Bei diesen Leuten erfordern stark konsistente Schreibvorgänge das Schreiben in ein Quorum von Shards, wodurch Sie mehr Geld kosten.

[3] `https://aws.amazon.com/de/dynamodb/faqs/What_is_a_readwrite_capacity_unit`
[4] `https://docs.datastax.com/en/cassandra/3.0/cassandra/dml/dmlConfigConsistency.html`

2.3.2 Operational Transformation

siehe woot OT ist eine weit verbreitete Technologie zur Unterstützung von Funktionalitäten in Kollaborativer Software. Sie stammt aus einer im Jahre 1989 veröffentlichten Forschungsarbeit und wurde ursprünglich nur für die gemeinsame Bearbeitung von Klartext-Dokumenten entwickelt [EG89]. Später ermöglichte weitere Forschung OT durch Unterstützung von *Sperrungen*, *Konfliktlösungen*, *Benachrichtigungen*, *Bearbeitung von Baumstrukturierten Dokumenten*, ... zu verbessern und erweitern. Im Jahr 2009: Google Wave, Google Docs
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 Latenz verursacht wird. Gleichzeitig auftretende Mehrfachänderungen sollen nicht zu unterschiedlichen Dokumentenzuständen führen.

Funktionsweise

Kollaborative Systeme, die OT verwenden, benutzen normalerweise den replizierten Dokumentenspeicher. Das heißt jeder **Client** verfügt über eine eigene Kopie des Dokuments. 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. Die Operationen erfolgen auf lokalen Kopie und die Änderungen werdenn an alle anderen **Clients** weitergegeben. Wenn ein **Client** die Änderungen von einem anderen **Client** empfängt, werden die Änderungen normalerweise **vor** ihrer Ausführung transformiert. Die Transformation stellt sicher, dass anwendungsabhängige Konsistenzkriterien (Invarianten) von allen Standorten gepflegt werden.
Es gibt die Operationen **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 **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 `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.
Des Weiteren unterstützt OT Operationen wie `update`, `point`, `lock`.

Beispiel : Benutzer A fügt an Position 12 das Zeichen 'A' ein Benutzer B fügt am Anfang des Dokuments ein 'B' ein. 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. Die Grundidee von OT besteht darin, die Parameter einer Editieroperation gemäß den Auswirkungen **zuvor ausgeführter** konkurrierender Operationen anzupassen, so dass die transformierte Operation die korrekte Wirkung erzielen und die Dokumentenkonsistenz aufrechterhalten kann.

STRUKTUR?

Transformations- 1. Kontrolle, 2. Eigenschaften, Bedingungen, 3. Funktionen
Vorteile der Trennung?

KRITIK

False-Tie puzzle? *A Generic Operation Transformation Scheme for Consistency Maintenance in Real-time Cooperative Editing Systems* und *Achieving convergence, causality-preservation, and intention-preservation in real-time cooperative editing systems*

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: “Aufgrund 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: “Einen 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].

KONSITENZMODELLE??? CC MODELL, CCI, CSM, CA

2.3.3 CONFLICT-FREE REPLICATED DATA TYPE

Die Idee von CRDTs ist, dass jeder "Typ"(wie ein Einkaufswagen) mit Intelligenz handelt, um Konflikte automatisch zu lösen.

CRDTs sind Objekte, die ohne teure Synchronisation aktualisiert werden können. Sie konvergieren schließlich, wenn alle gleichzeitigen Aktualisierungen kommutativ[5] 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.

OPERATIONSBASIERTER ANSATZ (OT?)

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**

[5] Unverändert bei Vertauschen der Operanden

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.

2.4 Das CouchDB Replikationsmodell

Aufgabe der Replikation von CouchDB ist die Synchronisation 2+n Datenbanken. Lösungen: Zuverlässige **Synchronisation** von Datenbanken auf verschiedenen Geräten. **Verteilung** der Daten über ein Cluster von DB-Instanzen die jeweils einen Teil des requests beantworten (Lastverteilung) und **Spiegelung** der Daten über geografisch weit verteilte Standorte.
Durch die inkrementelle (schrittweise) Arbeitsweise kann CouchDB genau dort weitermachen wo es unterbrochen wurde wenn während der Replikation ein Fehler auftritt, beispielsweise durch eine ausfallende Netzwerkverbindung *Es werden auch nur die Daten übertragen, die notwendig sind, um die Datenbanken zu synchronisieren.*
Das Besondere an CouchDB ist, dass es darauf ausgerichtet ist, Fehler/Konflikte vernünftig zu behandeln statt anznehmen es träten keine auf (vgl. [ALS10] S. 7f). Wie oben beschrieben, gibt es in Verteilten Systemen einige Fehler die auftreten können.

Das CouchDB Replikationsmodell erlaubt eine nahtlose, peer-to-peer (direkte) Datensynchronisation zwischen beliebig vielen Geräten. Das CouchDB Replikationsprotokoll ist in CouchDB selbst implementiert, das die Serverkomponente abdeckt. Dann gibt es das PouchDB-Projekt, das dasselbe Protokoll in JavaScript implementiert, das auf Browser- und Node.js-Anwendungen abzielt. das deckt Ihre Kunden und dev-Server ab. Schließlich gibt es Couchbase Mobile und Cloudant Sync, die auf iOS und Android laufen und das CouchDB Synchronisationsprotokoll in Objective-C bzw. Java implementieren.
Vektoruhr [6]
content addressable versions: Idee: Nimm den Objektinhalt (content) und jag ihn durch eine Hashfunktion

2.4.1 SCHLUSSENDLICHE KONSISTENZ

LOKALE KONSISTENZ

VERTEILTE KONSISTENZ

2.4.2 REPLIKATION?

2.4.3 KONFLIKTMANAGEMENT

[6] `https://en.wikipedia.org/wiki/Vector_clock`

3 Bestehende offlinefähige Systeme / Konzepte

Harte, weiche, mittlere probleme (verschiedene Stufen von offlinefähig) Native Apps sind offlinefähig
bei nativen Apps ist das Problem bei der Datenverteilung

3.1 Kollaborative Software

rauslassen? Kann ich benutzen um kollaborativ zu arbeiten –> **was passiert wenn ich offline bin?**

3.1.1 Google Docs

(benutzt OT)

3.1.2 Google Wave

(benutzt OT)

3.1.3 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)
–> 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)

plus wachsende Anzahl von offen zur Verfügung gestellter Bibliotheken und Frameworks die es ermöglichen, offlinefähige Anwendungen zu programmieren. Siehe Kapitel 3.2

3.2 Offline-First Frameworks/Bibliotheken

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. Progressive Web App

3.2.1 Realm

Backend für mobile Anwendungen (Java, Swift, C#, JS). Realm Datenbank oder Realm Platform(= DB+ Object Server). Schreiben groß 'OIffline First Experience' überall hin (webseite, whitepaper...)

- lokale DB, plattformübergreifend
- Object Server fungiert als Middleware-Komponente in der mobilen Applikation (App)-Architektur und managed die Datensynchronisation, eventhandling und Integration in Legacy-Systeme. Kann Daten effizient und simultan synchronisieren und **löst in Echtzeit, automatisch Konflikte**
- **Key-features:** Datensynchronisation in Echtzeit, Skalierbarkeit, Cross-Platform Datenmodell, Eventhandling, regelmäßige Backups, Datenintegrations API, Datensicherheit
- **key mobile use cases:** Reactive app architectures, Offline-first experiences, mobilizing legacy apis, realtime collaboration

[rea17b]

- Realtime Data Sync (sendet automatisch Änderungen in Echtzeit)
- Daten-sync-Protokoll komprimiert die marginalen Änderungen (statt das ganze Objekt) in Binärformat und übergibt sie zwischen Gerät und Server.
- synchronisiert die spezifischen Operationen zusammen mit den Daten
- Diese zusätzlichen Informationen erfassen genau das, was man beabsichtigt hat, **sodass das System Konflikte automatisch auflösen kann**. Dies führt zu einer vorhersagbaren Synchronisation ohne manuellen Eingriff, der die Leistung beeinträchtigt.

- (Objektorientierte) Datenbank auf dem Gerät
- Echtzeit Synchronisation
- Konfliktlösung benutzt OT (und vorgegebene Regeln): Man kann custom Konfliktlösungs-Regeln erstellen
- Unterstützung von Transaktionen? Ist das nicht normal? – Konfliktlösung passiert auf **Transaktionsebene**
- Wenn Änderungen aufgrund einer unterbrochenen Netzwerkverbindung offline gehen oder das Gerät leer ist, gehen keine Daten verloren.

[rea17a]

3.2.2 REDUX OFFLINE

"Persistenter Redux store für *reasonaboutable*™ Offline-First Anwendungen".
Ist ein eigenständiger Statuscontainer und kann mit jeder Webanwendung angewandt werden, die sich *deklarativ auf Basis einer einzigen Datenquelle rendern lässt*, wie beispielsweise React[7], Vue[8], oder Angular[9] [reda].
ist eine experimentelle Bibliothek die auf *battle-tested* patterns der Offline-First Architektur aufbaut
REDUX OFFLINE verspricht nicht, die Webanwendung offlinefähig zu machen. Um *assets*(Bilder, Skripte etc) zwischenzuspeichern, muss zusätzlich noch ein ServiceWorker implementiert sein.
Benutzt redux-persist und redux-optimist.
Bei jeder Änderung wird der Redux store auf dem Datenträger gespeichert, und bei jedem Start automatisch neugeladen. (Standardmäßig IndexedDB, localForage, AsyncStorage)
Eine mit REDUX OFFLINE erstellte Anwendung funktioniert ohne weitere Anpassung offline im Lesemodus. Also wenn die benutzende Person vom (Redux-)Status lesen möchte. Um auch im Schreibmodus offline zu funktionieren, werden alle Netzwerkgebundenen Aktionen in einem Store-internem Queue gespeichert. Dann erstellt REDUX OFFLINE einen Unterbaum `offline`, wo unter anderem der internen Status und ein Array namens `outbox` verwaltet wird. Um diese Aktivitäten bei Internetverbindung ausführen zu können, müssen alle notwendigen Daten Plus Metadaten gespeichert werden. Die Metadaten sind für die Information zuständig, was davor oder danach passieren soll. Es gibt drei Metadaten die REDUX OFFLINE interpretieren kann:
`meta.offline.effect` - Die Daten die gesendet werden sollen?

[7] JavaScript Bibliothek: `https://reactjs.org/`
[8] JavaScript Framework: `https://vuejs.org/`
[9] JavaScript Framework: `https://www.angular.io`

`meta.offline.commit` - Aktion die ausgeführt wird sobald Daten erfolgreich gesendet wurden
`meta.offline.rollback` - Aktion die bei permanent fehlgeschlagener Internetverbindung [redb]

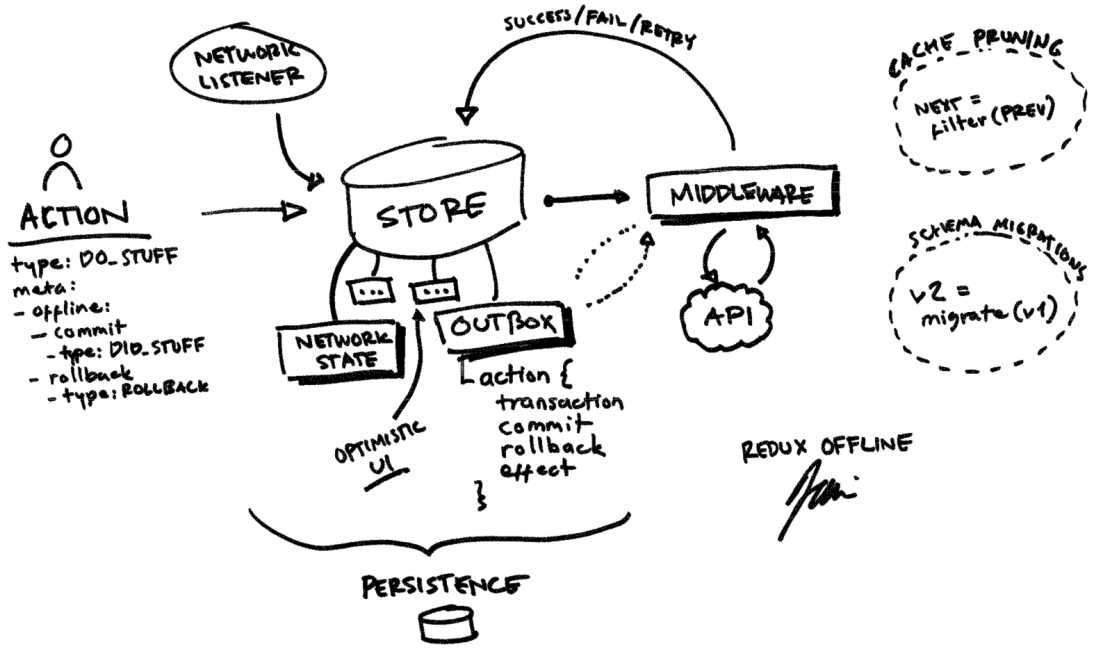

Abbildung 3.1: Redux Offline Architektur Quelle: [Evä17]

Die grundlegende Idee hinter Redux Offline ist, dass der REDUX STORE die Datenbank ersetzt/ist. Jede Aktion die benötigt wird um offline zu arbeiten, wird im STORE persisiert und durch die `meta.offline` -Daten weiß die Anwendung was online zu tun ist. Wie die Grafik 3.1 (links) zeigt, wird jede (offline-unterstützende) Aktion mit dem `offline.meta` Feld dekoriert. Darin wird beschrieben, wie der *Netzwerkeffekt* ausgeführt werden soll (`effect`) und welche Aktion ausgelöst werden soll, wenn sie erfolgreich (`commit`) ist oder fehlschlägt (`rollback`). Diese Offline-Aktionen werden im STORE-internen Queue gespeichert und werden, einmal online, an den Server gesendet.

User Interface (UI)
Es umfasst netzwerkfähige Application Programming Interface (API)-Aufrufe, das Persistieren des Zustands(Status), das Stapeln von Nachrichten und die Behandlung von Fehlern, Neuversuchen, optimistische Bedienoberfläche-Aktualisierungen, Migrationen, Cache-Bereinigung und mehr. Weil das kompliziert sein kann, zielt REDUX OFFLINE darauf ab, eine vernünftige Reihe von Standardverhaltensweisen zu bieten, die verwendet werden,

und nach Bedarf einzeln überschrieben werden können.
Grundsätzlich ist das Problem, das REDUX OFFLINE löst, kein technisches, sondern ein Architekturproblem. Architekturen können im Code implementiert werden, aber um verstanden, angewendet und nachvollziehbar zu werden, müssen sie gut kommuniziert werden. –> Dokumentation
Konflikte?

REDUX-PERSIST

localStorage. github [redc] medium [San16]

REDUX-OPTIMIST

3.2.3 REACT-NATIVE-OFFLINE

React Native = JavaScript React Framework um native, mobile Apps zu bauen. blabla
Behandelt online/offline Verbindung. Kann die Internetverbindung auch regelmäßig prüfen
Speichert nur den Status online/offline im store. -> erlaubt so unterschiedliches Rendern von Componenten.

```
const YourComponent = ( isConnected ) => (
<Text>isConnected ? 'I am connected to the internet!' : 'Offline :('</Text>
);
```

Zusammen mir Redux hat es einen 'Mehrwert': Hat dann einen 'Offline-Queue um Aktionen zu wiederholen (im Intervall) `meta.retry? Array of actions which, once dispatched, will trigger a dismissal from the queue` oder nicht `meta.dismiss? []`. [Acu]

3.2.4 OFFLINE-PLUGIN FÜR WEBPACK

webpack ist ein JavaScript 'Bundler', packt JavaScript-Dateien und oder assets für die Verwendung in Browsern.blabla
Bietet offline experience für webpack Projekte. Benutzt SERVICEWORKER und APPCACHE unter der Haube –> cached nur (gebündelten) von webpack generierten assets. Für die anderen Dateien (z.B. index.html die nicht gebundled wird oder Modul von CDN) braucht mal ein `html-plugin` oder benutzt die `externals` :

```
const OfflinePlugin = require('offline-plugin')
```

```
const offline = new OfflinePlugin

const offline = new OfflinePlugin(
externals: ['index.html'],
)
```

Es gibt diverse config-options für SERVICEWORKER und AppCache... [Sto]

3.2.5 HOODIE

Benutzt CouchDB und PouchDB... [hoo]

COUCHDB

Apache CouchDB™ist ein Datenbank Management System (DBMS) das seit 2005 als freie Software entwickelt wird. Die dokumentenorientierte Datenbank (DB) funktioniert sowohl als einzelne Instanz, als auch im Cluster, in dem ein Datenbanksserver auf einer beliebig großen Anzahl an Servern oder Virtuelle Maschines (VMs) ausgeführt werden kann. So kann die Datenschicht beliebig skaliert werden, um die Anforderungen vieler BenutzerInnen zu erfüllen. CouchDB verwendet das HTTP–Protokoll und JavaScript Object Notation (JSON) als Datenformat, weswegen es mit jeder Software kompatibel ist, die das unterstützt.
Eine weitere Besonderheit ist das implementierte Replikationsmodell. Dessen Funktionsweise wird in Abschnitt 2.4 detailliert beschrieben. Dieses Protokoll ist die Grundlage für Offline First Anwendungen. So kann beispielsweise eine CouchDB-Instanz auf dem Mobiltelefon und eine auf dem Laptop bestehen und beide können sich bei bestehender Internetverbindung synchronisieren. Da so die gespeicherten Daten aus dem lokalen Speicher gelesen werden, sind ein schnelles Interface und eine geringe Latenz die positive Folge. Wenn Konflikte auftreten, beispielsweise durch gleichzeitiges Bearbeiten eines Dokuments von zwei Personen ohne Netzwerkverbindung, gehen keine Daten verloren und es liegt an der benutzenden Person diese zu lösen. CouchDB ist für Server konzipiert. Für Browser gibt es PouchDB und für native iOS- und Android–Apps wurde Couchbase Lite entwickelt. Alle können Daten miteinander replizieren und verwenden das CouchDB Replikationsprotokoll. [cou]

Replikation?

Eventual Consistency?

PouchDB

Als Ergänzung zu CouchDB kann PouchDB verwendet werden. PouchDB ist eine Open–Source–JavaScript–Datenbank, die so konzipiert wurde, dass sie im Browser läuft. PouchDB ermöglicht es Anwendungen zu erstellen, die sowohl offline als auch online funktionieren. Daten können lokal gespeichert werden, sodass alle Funktionen der Anwendung auch im Offline–Modus zur Verfügung stehen. Daten werden unabhängig von der nächsten Anmeldung (des nächsten Onlinezugangs) zwischen **Clients**, CouchDB oder kompatiblen Servern synchronisiert. PouchDB läuft auch in Node.js[10] und kann als direkte Schnittstelle zu CouchDB–kompatiblen Servern verwendet werden [pou].

3.2.6 Couchbase

[10] JavaScript Laufzeitumgebung, steht unter `https://nodejs.org/en/download/` zum Download bereit

4 SZENARIEN ...

Alle in Kapitel 3 angeführten ... haben die Gemeinsamkeit...
Prinzipiell sollte die Anwendung in der Lage sein, ..., die sich aus den Szenarien ergeben.
Folgende Situationen können eintreten:
langsames Internet, 3G LTE, Timeout während Request

Szenario R1:
blabla

Szenario R2:
blabla

Szenario R3:
blabla

Szenario G1:
blabla.

Szenario G2:
blabla.

Szenario G3:
blabla.

ERGEBNIS

Da die Szenarien *R2* und *G2*, die Szenarien *R3* und *G3* sowie die Szenazien*V1* und *V2* zusammengefasst werden können, ergeben sich aus den acht Szenarien im Ampelbereich die fünf nun aufgezählten Fälle.

- Fall a: blabla
- Fall b: blabla
- Fall c: blabla

- Fall d: blabla
- Fall e: blabla

Im weiteren Verlauf dieser Arbeit wird unter anderem beschrieben, wie diese Fälle in die Konzeption der zu *entwickelnden Anwendung* eingebunden werden.

5 Anforderungsdefinition

Dieses Kapitel beschreibt die Anforderungen an ... unter Berücksichtigung von Funktionalität und .... Heutzutage spielt die Benutzbarkeit und Funktionalität des Produkts eine große Rolle.
Aus den oben genannten Szenarien werden im Folgenden die Anforderungen hergeleitet, die ... erfüllen soll. Für die Umsetzung einer solchen Applikation ist ...obligatorisch.

5.1 Anwendungsfälle

Aus den in Kapitel 4 erarbeiteten Szenarien ergeben sich die folgenden **drei** Use-Cases, die von der Anwendung erfüllt werden sollen.

ID	Anwendungsfall	Beschreibung
UC1	Ich.., um ...	Es passiert das und das.
UC2	Ich.., um ...	Es passiert das und das.
UC3	Ich ..., um d...	Es passiert das und das.

Tabelle 5.1: Anwendungsfälle

Dann Use-Case-Diagramm

5.2 Funktionalität

5.3 Die graphische Oberfläche

Optimistische Bedienoberfläche UI soll mich nicht mit Meldungen darüber nerven, dass ich offline bin. (Bsp. Chat)
UI soll sagen wenn es einen Konflikt gab / gibt und mich entscheiden lassen. Bzw ihn lösen

lassen. Auf keinen Fall selber lösen und mich nichts davon wissen lassen.
Im besten Fall soll due UI mir sagen **warum** es zum Konflikt gekommen ist.

6 Vorgehen

- was verspricht redux-offline
- Wie funktioniert redux-offline? Welche Strategie zur Konfliktlösung wird verwendet?
- was könte daran problematisch sein?
- Wie hoch ist der Implementierungsaufwand?
- Dasselbe für Couch & Pouch

Implementierungsaufwand: Anforderungen an Simulator? Konzeption? usw.

Abkürzungen

API	Application Programming Interface
App	Applikation
CAP	Consistency Availability Partition tolerance
CRDT	Conflict-free replicated data type
DB	Datenbank
DBMS	Datenbank Management System
JSON	JavaScript Object Notation
LWW	Last-Write-Wins
OT	Operational Transformation
PWA	Progressive Web App
UI	User Interface
VM	Virtuelle Maschine
WLAN	Wireless Local Area Network

Glossar

Bandbreite
: gibt an, wie viele Daten pro festgelegter Zeitspanne über ein Netzwerk übertragen werden können.

Hashfunktion
: TODO: ist eine Abbildung, die eine große Eingabemenge (die Schlüssel) auf eine kleinere Zielmenge (die Hashwerte) abbildet

Kollaborativ
: Als kollaborative Software oder kollaboratives System wird eine Software zur Unterstützung der computergestützten Zusammenarbeit in einer Gruppe über zeitliche und/oder räumliche Distanz hinweg bezeichnet

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

Middleware
: Schicht zwischen Anwendung und Betriebssystem.

optimistische Bedienoberfläche
: auch: optimistic UI, wartet nicht mit der Aktualisierung der Oberfläche auf das Ende einer Operation. Die zeigt also den gewünschten Zustand der App an, befor die Anwendung fertig ist indem sie z.B. Fakedaten zeigt.

Progressive Web App
: Oder “fortschrittliche Web App“ eine mobil nutzbare Webseite, erstellt mit den Webstandards, die als Symbiose aus einer nativen, mobilen Anwendung und einer responsiven Webseite beschrieben werden kann. Die Idee dahinter ist, dass Apps zukünftig nicht mehr über einen App Store, sondern über den Browser installiert werden kann.

Queue
: auch Warteschlange, eine Datenstruktur die zur Zwischenspeicherung von Objekten dient. Hierbei wird das zuerst eingegebene Objekt auch zuerst verarbeitet (wie bei einer Warteschlange).

ABBILDUNGSVERZEICHNIS

LITERATURVERZEICHNIS

[Acu] ACUÑA, Raúl G.: *react-native-offline*. https://github.com/rauliyohmc/react-native-offline, . – Zugriff: 12.04.2018 3.2.3

[ALS10] ANDERSON, J. C. ; LENHARDT, Jan ; SLATER, Noah: *CouchDB: The Definitive Guide*. Sebastopol, CA : O'Reilly Media, 2010. – ISBN 978–0–596–15589–6. – oreilly.com 2.4

[Ban16] BANK, World: World Development Report 2016: Digital Dividends / International Bank for Reconstruction and Development / The World Bank. Washington DC, 2016. – Research Report. – doi: doi:10.1596/978-1-4648-0671-1 1

[cou] *CouchDB – relax*. https://couchdb.apache.org/, . – Zugriff: 12.04.2018 3.2.5

[EG89] ELLIS, C. A. ; GIBBS, S. J.: Concurrency Control in Groupware Systems. In: *SIGMOD Rec.* 18 (1989), jun, Nr. 2, 399–407. http://dx.doi.org/10.1145/66926.66963. – DOI 10.1145/66926.66963. – ISSN 0163–5808 2.3.2

[Evä17] EVÄKALLADO, Jani: Introducing Redux Offline: Offline-First Architecture for Progressive Web Applications and React Native. In: *HACKERnoon* (2017), 3. – Zugriff: 12.04.2018 3.1

[fal] *Eight Fallacies of Distributed Computing*. https://blog.fogcreek.com/eight-fallacies-of-distributed-computing-tech-talk/, . – Zugriff: 12.04.2018 2.2

[Fra09] FRASER, Neil: Differential Synchronization. Version: Jan 2009. https://neil.fraser.name/writing/sync/eng047-fraser.pdf. 2009. – Research Report. – 8 S. 2.3.2

[hoo] *Hoodie – The Offline First Backend*. http://hood.ie, . – Zugriff: 12.04.2018 3.2.5

[LL10] LI, Du ; LI, Rui: An Admissibility-Based Operational Transformation Framework for Collaborative Editing Systems. In: *Comput. Supported Coop. Work* 19

(2010), feb, Nr. 1, 1–43. http://dx.doi.org/10.1007/s10606-009-9103-1. – DOI 10.1007/s10606–009–9103–1. – ISSN 0925–9724 2.3.2

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

[pou] *pouchdb – Tha Database that Syncs!* https://pouchdb.com/learn,. – Zugriff: 12.04.2018 3.2.5

[rea17a] *The Offline-First Approach to Mobile App Development - Beyond Caching to a Full Data Sync Platform*. https://www2.realm.io/whitepaper/offline-first-approach-registration, october 2017. – Zugriff: 12.04.2018 3.2.1

[rea17b] *BUILD BETTER APPS, FASTER WITH REALM - An Overview of the Realm Platform*. https://www2.realm.io/whitepaper/realm-overview-registration, october 2017. – Zugriff: 12.04.2018 3.2.1

[reda] *Redux Offline Release BREAKING: Migrate to Store Enhancer API*. https://github.com/redux-offline/redux-offline/releases/tag/v2.0.0, . – Zugriff: 12.04.2018 3.2.2

[redb] *Redux Offline*. https://github.com/redux-offline/redux-offline,. – Zugriff: 12.04.2018 3.2.2

[redc] *react-native-offline*. https://github.com/rt2zz/redux-persist,. – Zugriff: 12.04.2018 3.2.2

[San16] SANFORD, Clark: Persistence is Key: Using Redux-Persist to Store Your State in LocalStorage. In: *Medium* (2016), 12. – 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: Jan 2011. https://hal.inria.fr/inria-00555588. 2011 (RR-7506). – Research Report. – 50 S. 2.3.3

[SPBZ11b] SHAPIRO, Marc ; PREGUIÇA, Nuno ; BAQUERO, Carlos ; ZAWIRSKI, Marek: Conflict-free Replicated Data Types. Version: Jul 2011. https://hal.inria.fr/inria-00609399. 2011 (RR-7687). – Research Report. – 18 S. 2.3.3, 2.3.3

[Sto] STOLYAR, Arthur: *offline-plugin*. https://github.com/NekR/offline-plugin,. – Zugriff: 12.04.2018 3.2.4

Anhang

Eidesstattliche Erklärung

CD-Inhalt

Auf der beigefügten CD befinden sich

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