tgm
Die Schule der Technik

JUST DO IT
HÖHERE ABTEILUNG FÜR
INFORMATIONSTECHNOLOGIE

Systemtechnik Labor

5BHIT 2017/18

Synchronisation - Einkaufsliste

Theorie

Kevin Waldock

18. April 2018

Bewertung:		Version:	0.1
Betreuer:	Michael Borko	Begonnen:	13.4.18
		Beendet:	15.4.18

Inhaltsverzeichnis

1 Einführung **3**
1.1 Ziele . . . 3
1.2 Voraussetzungen . . . 3
1.3 Aufgabenstellung . . . 3
1.4 Bewertung . . . 3

2 Umsetzung **5**
2.1 Voraussetzungen . . . 5
2.2 Synchronisationsalgorithmus . . . 5
2.2.1 Herausforderungen . . . 5
2.2.2 Sperrbasiert/Wechselweise . . . 5
2.2.3 Änderungsbasierte Synchronisation . . . 6
2.3 diffsync . . . 11
2.4 Die Replikation . . . 11
2.5 Server . . . 11
2.6 Client . . . 11
2.6.1 Funktionsweise . . . 11
2.7 Ausführung . . . 12
2.7.1 Server . . . 12
2.7.2 Client . . . 12

Literaturverzeichnis **13**

1 Einführung

Diese Übung soll die möglichen Synchronisationsmechanismen bei mobilen Applikationen aufzeigen.

1.1 Ziele

Das Ziel dieser Übung ist eine Anbindung einer mobilen Applikation an ein Webservices zur gleichzeitigen Bearbeitung von bereitgestellten Informationen.

1.2 Voraussetzungen

- Grundlagen einer höheren Programmiersprache
- Grundlagen über Synchronisation und Replikation
- Grundlegendes Verständnis über Entwicklungs- und Simulationsumgebungen
- Verständnis von Webservices

1.3 Aufgabenstellung

Es ist eine mobile Anwendung zu implementieren, die einen Informationsabgleich von verschiedenen Clients ermöglicht. Dabei ist ein synchronisierter Zugriff zu realisieren. Als Beispielimplementierung soll eine Ëinkaufsliste"gewählt werden. Dabei soll sichergestellt werden, dass die Information auch im Offline-Modus abgerufen werden kann, zum Beispiel durch eine lokale Client-Datenbank.

Es ist freigestellt, welche mobile Implementierungsumgebung dafür gewählt wird. Wichtig ist dabei die Dokumentation der Vorgehensweise und des Designs. Es empfiehlt sich, die im Unterricht vorgestellten Methoden sowie Argumente (pros/cons) für das Design zu dokumentieren.

1.4 Bewertung

- Gruppengrösse: 1 Person
- Anforderungen “Grundkompetenz überwiegend erfüllt”
  - Beschreibung des Synchronisationsansatzes und Design der gewählten Architektur (Interaktion, Datenhaltung)
  - Recherche möglicher Systeme bzw. Frameworks zur Synchronisation und Replikation der Daten
  - Dokumentation der gewählten Schnittstellen
- Anforderungen “**Grundkompetenz zur Gänze erfüllt**”
  - Implementierung der gewählten Umgebung auf lokalem System
  - Überprüfung der funktionalen Anforderungen zur Erstellung und Synchronisation der Datensätze
- Anforderungen “**Erweiterte-Kompetenz überwiegend erfüllt**”
  - CRUD Implementierung

  - Implementierung eines Replikationsansatzes zur Konsistenzwahrung

- Anforderungen "**Erweiterte-Kompetenz zur Gänze erfüllt**"
  - Offline-Verfügbarkeit
  - System global erreichbar

2 Umsetzung

Die Umsetzung wurde mit diffsync gemacht.

2.1 Voraussetzungen

Für das Testen der Umsetzung ist folgendes Vorausgesetzt:

- Node.js 8 LTS

2.2 Synchronisationsalgorithmus

Die Datensynchronisation in Echtzeit bei verteilten Systemen ist eine Herausforderung, da die Daten von verschiedenen Usern immer konsistent gehalten werden müssen. Für die Echtzeit muss die Synchronisation sehr häufig durchgeführt werden. Es gibt verschiedene Methoden und Algorithmen, um dies zu erzielen.

2.2.1 Herausforderungen

Der Synchronisationsprozess muss möglichst oft durchgeführt werden. Benötigt der Client zu viel Zeit, um mit dem Server zu synchronisieren, ist die Wahrscheinlichkeit für einen Zusammenführungskonflikt viel höher. Außerdem muss die Version des Servers und die des Clients konsistent bleiben. In verteilten Systemen ist dies nicht möglich, außer der Datenbestand wird blockiert, bis der andere Client die Änderung durchgeführt hat. In NoSQL-Systemen wird mit schlussendlicher Konsistenz gearbeitet. Dieses Modell verlangt nicht, dass der Datenbestand sofort mit der Gegenseite übereinstimmt, muss jedoch möglichst schnell einen konsistenten Zustand erreichen. [3]

Dennoch kann es zu Konflikten kommen. Wenn mehrere User an derselben Stelle eines Dokuments eine Änderungen durchführen, kommt es zu Problemen bei der Zusammenführung. Bei Synchronisationsprozessen, die nicht in Echtzeit durchgeführt werden, wird der User um eine manuelle Zusammenführung gebeten. In einem System, in der die Echtzeit wichtig und die Zusammenführung sehr häufig durchgeführt wird, ist eine manuelle Zusammenführung nicht möglich.

Auch die Möglichkeit eines Paketverlustes muss beachtet werden, wenn in einem verteilten System die Verbindung nicht zuverlässig ist. Der Client oder der Server hat keine Möglichkeit zu erkennen, ob die Gegenseite dessen Änderungen eingefügt hat. Im schlimmsten Fall arbeiten der Client und der Server mit unterschiedlichen Versionen, und dadurch ist die Konsistenz nicht gewährleistet. [3]

Es existieren verschiedene etablierte Synchronisationsmethoden, welche jedoch nicht für jede Situation geeignet sind. Abhängig von der Methode wird entweder die Konsistenz oder die Echtzeit priorisiert.

2.2.2 Sperrbasiert/Wechselweise

Bei dieser Methode wird nur ein User für das Editieren eines Dokuments zugelassen. Während dieser User den vollen Schreibzugriff hat, müssen andere warten, bis der User seine Änderungen durchgeführt hat. Die anderen User haben während der Dauer des Editierungsvorgangs nur Lesezugriff. [1]

Der Nachteil dieser Methode ist, dass nur ein User das Dokument editieren kann, und daher die Methode für kollaboratives Editieren nicht geeignet ist.

2.2.3 Änderungsbasierte Synchronisation

Bei der änderungsbasierten Synchronisation wird dem jeweiligen anderen Client der Änderungsbefehl mitgeteilt. Diese Methode ist sehr effizient, da nur sehr wenig Information übertragen werden muss. Jedoch müssen alle Änderungen beachtet werden. Darunter fallen Befehle wie einfache Editierungen, aber auch Kopieren, Einfügen, Ersetzen und die Autokorrektur.

Arbeiten mehrere User an einem Dokument, kann es zu Konflikten kommen, wenn gleichzeitig Änderungen durchgeführt werden. Wenn Client A an einer Textpassage eine Änderung durchführt und Client B an der gleichen Textpassage eine andere Änderung durchführt, dann werden diese an den Server geschickt. Der Server sendet dann die jeweiligen Änderungsbefehle an den jeweiligen anderen User.

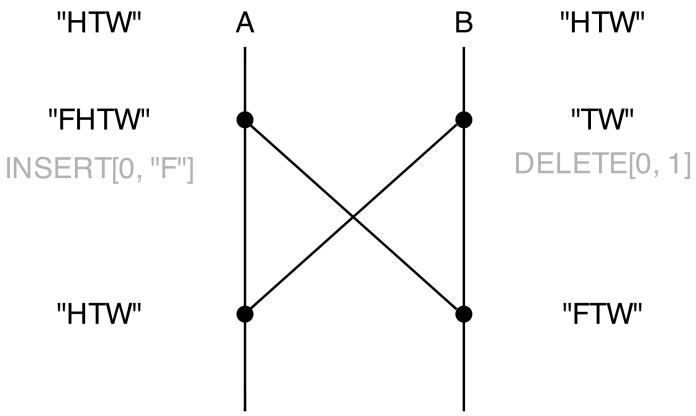

Abbildung 1: Probleme mit änderungsbasierter Synchronisation [2]

Sollte es zu so einem Fall kommen, dann arbeiten die verschiedenen Clients mit unterschiedlichen Versionen des Dokuments. Die Clients haben keine Möglichkeit, diese Inkonsistenz zu erkennen.

Weiterhin kann es zu einer Inkonsistenz kommen, wenn Änderungsbefehle über das Netzwerk verloren gehen. Der Client A nimmt an, dass die Änderungen bei Client B durchgeführt wurden, obwohl dies nicht der Fall ist. Auch hier kann die Inkonsistenz nicht erkannt werden.

Dreiweg-Zusammenführung

Die Dreiweg-Zusammenführung ist sehr bekannt, da sie in Versionierungssoftware wie Git oder SVN eingesetzt wird. Bei der Dreiweg-Zusammenführung ist eine Basisversion vorhanden, von der jeder Client ausgeht. Machen zwei Clients gleichzeitig eine Änderung, kann dies durch die Dreiweg-Zusammenführung gelöst werden.

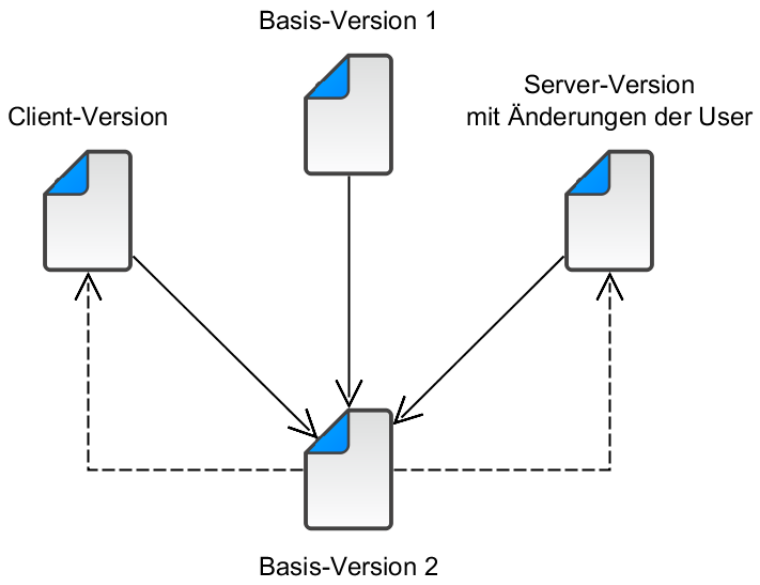

Abbildung 2: Dreiweg-Zusammenführung

Die Client-Version und die Server-Version gehen von einer Basis-Version aus. Diese wird benötigt, um von beiden Seiten (Server und Client) die Änderungen zusammenzuführen.

Die Dreiweg-Zusammenführung bietet die Möglichkeit, dass gleichzeitig mehrere Personen an einem Dokument arbeiten können. Jedoch dürfen während der Phase der Zusammenführung keine Änderungen vorgenommen werden, da die Basis-Version immer gleich sein muss. Daher ist diese Methode nicht für Echtzeit-Anwendungen geeignet.

Differentialsynchronisation

Die Differentialsynchronisation ist ein Synchronisationsalgorithmus, der von Neil Fraser entwickelt wurde. [1] Bei der Differentialsynchronisation arbeiten der Server und der Client jeweils mit einem Schattendokument. Dabei besitzt der Server **pro User** ein eigenes Schattendokument.

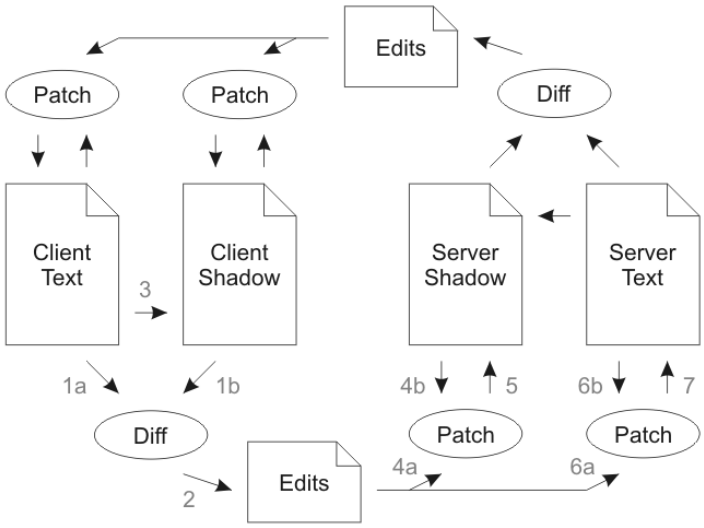

Abbildung 3: Differentialsynchronisation mit doppeltem Schattendokument [1]

Nach jeder Änderung wird versucht, das Dokument mit dem Server zu synchronisieren.

1. Der veränderte Text des Clients wird mit dem Schattendokument verglichen.
2. Die Liste an Änderungen wird gesammelt.
3. Der veränderte Text wird zum Schattendokument.
4. Die Veränderungen vom Client werden mit dem Schattendokument auf der Server-Seite zusammengefügt.
5. Das zusammengeführte Dokument ergibt das neue Schattendokument.
6. Die Veränderungen vom Client werden mit dem Dokument des Servers zusammengeführt.
7. Das zusammengeführte Dokument ergibt das neue Server-Dokument.

Es darf immer nur ein Netzwerk-Paket versendet werden. Das heißt, dass, wenn der Client ein Paket sendet, der Server keines senden darf, bis dieses Paket angekommen ist. Im obigen Modell (Abbildung 3) ist es jedoch nicht möglich, dies zu erkennen. Daher müssen sich der Client und der Server mit dem Senden und Empfangen abwechseln, damit es nicht zu Konflikten kommt.

Der gleiche Prozess kann auch in die andere Richtung durchgeführt werden. Es ist essentiell, dass das Schattendokument der Client-Seite und das auf der Server-Seite gleich sind. Dabei können Prüfsummen helfen sicherzustellen, dass die Dokumente übereinstimmen. Die Schattendokumente sind die Basis für richtige Patches. Wenn nun ein Paket verloren geht, dann hat der Server keine Möglichkeit, die Änderungen anzunehmen. Daraus folgt, dass die Schattendokumente auf der Client- und Server-Seite nicht gleich sind. Der Client würde bei einer weiteren Änderung mit einem falschen Schattendokument arbeiten.

Differentialsynchronisation mit garantierter Übermittlung
Dieses Modell basiert auf der Differentialsynchronisation mit dem Zusatz von Versionsnummern und einem Backup-System.

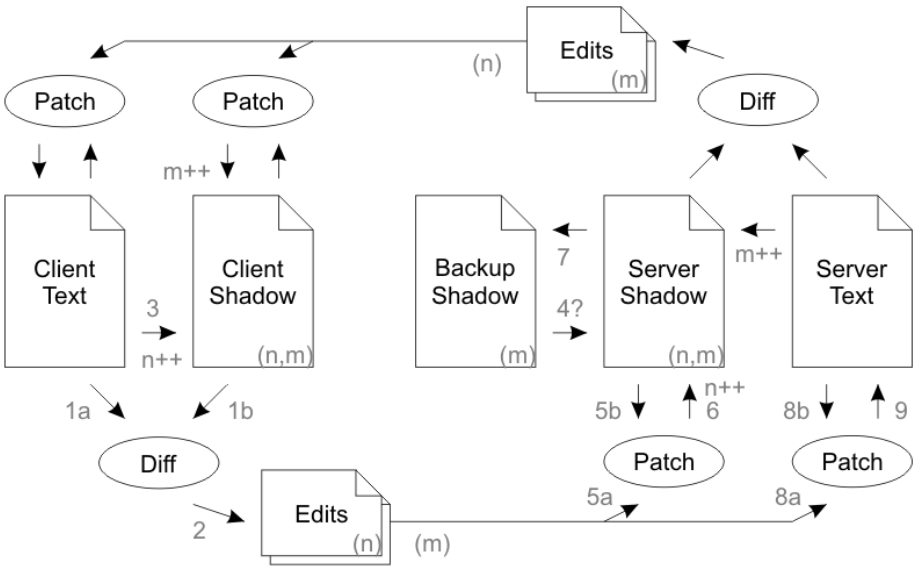

Abbildung 4: Differentialsynchronisation mit garantierter Übermittlung [1]

1. Der veränderte Text des Clients wird mit dem Schattendokument verglichen.
2. Änderungen werden in einer Liste gesammelt. Es handelt sich hierbei um eine Liste von Änderungen mit den jeweiligen Versionsnummern.
3. Der veränderte Text wird zum Schattendokument. Die Versionsnummer des Clients erhöht sich. Nach diesem Schritt werden die Veränderungen an den Server versendet. Dabei hat jede Veränderung ihre eigene Versionsnummer und eine gemeinsame Server-Versionsnummer. Die Liste an Änderungen wird erst dann gelöscht, wenn der Server dies auch bestätigt.
4. Die Server-Versionsnummer wird verglichen. Falls die Server-Versionsnummer vom Client älter sein sollte, wird das Backup-Schattendokument geladen.
5. Die Veränderungen vom Client werden mit dem Schattendokument auf der Server-Seite zusammengefügt.
6. Das zusammengeführte Dokument ergibt das neue Schattendokument. Die Client-Versionsnummer wird auf den neuesten Stand gesetzt.
7. Das momentane Schattendokument überschreibt das Backup-Schattendokument.
8. Die Veränderungen vom Client werden mit dem Dokument des Servers zusammengeführt.
9. Das zusammengeführte Dokument ergibt das neue Server-Dokument.

Auch hier kann der gleiche Prozess in der anderen Richtung durchgeführt werden, mit dem Unterschied, dass auf der Client-Seite kein Backup-Schattendokument existiert. Durch den Einsatz eines Backup-Schattendokuments können mögliche Probleme behandelt werden:

- **Doppeltes Paket:** In diesem Fall werden die Änderungen zweimal versendet. Die Versionsnummer ist dabei gleich. Beim ersten Mal werden diese Änderungen vom Server bearbeitet. Beim zweiten Mal erkennt der Server jedoch, dass seine Client-Versionsnummer höher ist. Die Änderungen werden ignoriert.
- **Verlust eines Client-Pakets:** In diesem Fall geht beim ersten Versenden das Netzwerk-Paket verloren, beim zweiten Mal jedoch nicht. Da die Liste an Änderungen erst bei einer Antwort des Servers gelöscht wird, kann der Client nach einer kurzen Zeit diese Liste wieder versenden. In dieser Zeit hat der Client auch die Möglichkeit, weitere Änderungen durchzuführen.
- **Verlust eines Server-Pakets:** In diesem Fall hat der Client seine Antwort nie bekommen. Der Client nimmt an, dass sein Client-Paket verloren ging. Er sendet deswegen nochmal ein Client-Paket. Der Server merkt, dass die Server-Versionsnummern nicht übereinstimmen. Daraufhin lädt der Server die alte Version vom Backup-Schattendokument und beginnt wieder von dieser Version aus seine Änderungen durchzuführen.

Wie auch bei der Dreiweg-Zusammenführung kann es in diesem Modell zu unlösbaren Zusammenführungskonflikten kommen. Im Unterschied zur Dreiweg-Zusammenführung verläuft die Differentialsynchronisation in einer Umgebung, in der die Echtzeit wichtig ist. Es besteht hier keine Möglichkeit, dass eine Person diese Konflikte manuell lösen kann. Falls es zu einem unlösbaren Zusammenführungskonflikt kommen würde, wird der Client gezwungen, auf den Stand des Servers zurückzukehren. Da die Synchronisation sehr häufig geschieht, sind diese Verluste minimal.

Skalierung und Replikation Die einfachste Möglichkeit, dieses System zu skalieren, ist es, die Anwendung auf verschiedenen Rechnern zu verteilen, die jeweils für eine bestimmte Anzahl an Dokumenten zuständig sind.

Sollte jedoch eine Skalierung auf ein einzelnes Dokument benötigt werden, dann teilen sich die jeweiligen Server die Benutzer. Die Differentialsynchronisation kann auch zwischen Server und Server angewendet werden.

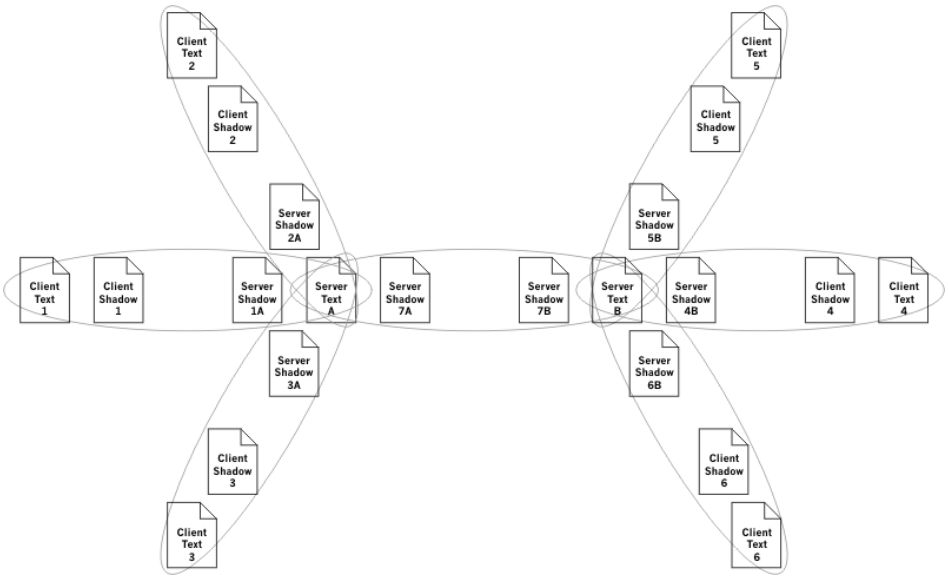

Abbildung 5: Differentialsynchronisation mit mehr als zwei Servern [1]

2.3 diffsync

Diffsync ist eine leightweight Library für die Differentialsynchronisation. Es bietet eine Adapter-Schnittstelle, welche für das jeweilige gewünschte Framework implementiert werden kann. Die Schnittstelle besteht aus den Methoden `getData()` und `storeData()`.

2.4 Die Replikation

Da diffsync unabhängig vom Datenbankmanagementsystem ist, kann jedes verwendet werden. Couchbase ist eine NoSQL-Datenbank, welche Replikationsmöglichkeiten bietet. So kann diffsync mit Couchbase integriert werden, wenn der Adapter implementiert wird.

2.5 Server

Der Server besteht lediglich aus der diffsync-Library und Socket.io welche auf Port 3000 lauscht. Diffsync kümmert sich um die Synchronisierung nach dem Differentialalgorithmus. Dank Socket.io ist auch eine automatische Wiederverbindung möglich.

2.6 Client

Der Client besteht aus der index.html und der Javascript-Datei index.js welches in ES6 geschrieben wurde und mit webpack in einer bundle.js transpiliert wird.

2.6.1 Funktionsweise

DOM-Modifikation Nach jeder Veränderung des data-Objekts wird das ganze DOM-Model neu gebildet. Dies bedeutet, dass alle DOM-Elemente gelöscht und neu hinzugefügt werden. Damit wird garantiert, dass das DOM-Model auch die Daten im data-Objekt repräsentiert. Außerdem wird ein sync-Befehl ausgeführt. Damit erhalten alle anderen verbundenen Clients die Daten und deren View wird auch aktualisiert.

Hinzufügen von Daten Wenn nun neue Daten hinzugefügt werden, dann wird das `data`-Objekt verändert. Die View wird aktualisiert und die Synchronisation wird ausgeführt.

Löschen von Daten Ein Datensatz kann mithilfe des "Löschen"-Buttons gelöscht werden. Der jeweilige Datensatz wird mithilfe der Position in der Liste identifiziert.

2.7 Ausführung

2.7.1 Server

Um den Server zu starten müssen folgende Schritte befolgt werden:

- In der Ordner des Server eintreten: `cd server`
- `npm install` ausführen, damit alle Abhängigkeiten installiert werden
- `npm start` ausführen, um den Websocket-Server zu starten.

2.7.2 Client

Um den Client zu testen müssen folgende Schritte befolgt werden:

- In der Ordner des Clients eintreten: `cd client`
- `npm install` ausführen, damit alle Abhängigkeiten installiert werden
- `npm start` ausführen damit der Client von webpack kompiliert wird.
- `npm run server` ausführen damit ein HTTP-Server auf 8080 gestartet wird, welche die Client-HTML-Datei ausgibt.

Literaturverzeichnis

[1] Neil Fraser. „Differential synchronization“. In: *Proceedings of the 9th ACM symposium on Document engineering*. ACM. 2009, S. 13–20. URL: https://neil.fraser.name/writing/sync/eng047-fraser.pdf (besucht am 25. 03. 2018).

[2] Jan Monschke. *diffsync - build collaborative applications easily*. 2017. URL: http://janmonschke.com/diffsync-presentation (besucht am 25. 03. 2018).

[3] A.S. Tanenbaum und M. Van Steen. *Verteilte Systeme: Prinzipien und Paradigmen*. Pearson Studium. Addison Wesley Verlag, 2008, S. 319–322. ISBN: 9783827372932. URL: http://books.google.at/books?id=V6I6PQAACAAJ.

Abbildungsverzeichnis

1 Probleme mit änderungsbasierter Synchronisation [2] 6
2 Dreiweg-Zusammenführung 7
3 Differentialsynchronisation mit doppeltem Schattendokument [1] 8
4 Differentialsynchronisation mit garantierter Übermittlung [1] 9
5 Differentialsynchronisation mit mehr als zwei Servern [1] 11