Evaluation von Synchronisations- und Konfliktlösungsverfahren im Web-Umfeld

Martin Eigenmann

19. Mai 2015

DRAFT

1

Abstract

2
Danksagungen

Thanks Mum.
Thanks Dad.
...

3

Inhaltsverzeichnis

Teil I.

Einleitung und Abgrenzung

4
Einleitung

4.1. Motivation und Fragestellung

Der Zugriff auf Services und Medien mittels mobiler Geräte steigt beständig an. So ist im Mai 2014, 60% der Zeit, die online verbracht wird, über Handy und Tablet zugegriffen worden - davon 51% mittels mobiler Applikationen. [4]

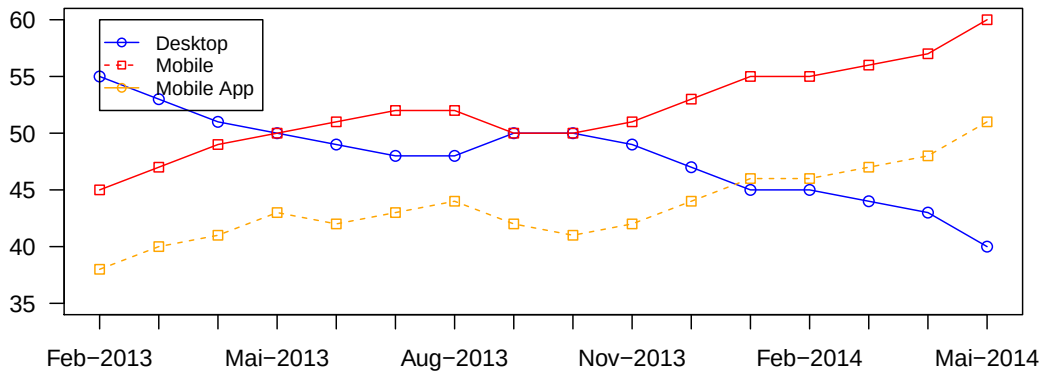

Abbildung 4.1.: Verteilung der online verbrachten Zeit nach Platform (Grafik erstellt gemäss der Daten von [4])

Angesichts der grossen Verbreitung und Nutzung von Services und Medien im Internet, wird eine unterbruchsfreie Benutzbarkeit, auch ohne Internetverbindung, immer selbstverständlicher und somit auch immer wichtiger.

> „It's clear that the mobile industry has finally given up on the fantasy that an Internet connection is available to all users at all times. Reality has set in. And in

> the past month, we've seen a new wave of products and services that help us go offline and still function.“ Elgan [3]

Es stellt sich nun die Frage, wie Informationen, über Verbindungsunterbrüche hinweg, integer gehalten werden können. Und wie Daten im mobilen Umfeld synchronisiert, aktualisiert und verwaltet werden könne, so dass für den Endbenutzer schlussendlich kein Unterschied zwischen Online- und Offline-Betrieb mehr wahrnehmbar ist.

4.2. Aufgabenstellung

Die von der Leitung des Studiengangs Informatik freigegebene Aufgabenstellung ist im Appendix unter „Aufgabenstellung“ aufgeführt.

4.3. Abgrenzung der Arbeit

4.4. Begründung

4.5. Geschichtliche Einordnung in das Thema

5

Dokumentationsstruktur und Beitrag zum Forschungsgebiet

Teil i - Einleitung und Abgrenzung

Teil ii - Technische Grundlagen und Architekturen

Teil iii - Konzept und Implementierung

Teil iv - Abschluss und Ausblick

Teil II.

Technische Grundlagen und Architekturen

6
Recherche

Dieses Kapitel erklärt die wichtigsten Grundbegriffe und wiedergibt die während der Recherche gesammelten Informationen.

6.1. Fachbegriffe

Eine Aufführung und Erlährung der Fachbegriffe befindet sinch im Appendix unter „Glossar“

6.2. Erläuterung der Grundlagen

6.2.1. Datenbanken

Eine Datenbank [1] ist ein System zur Verwaltung und Speicherung von strukturierten Daten. Erst durch den Kontext des Datenbankschemas wird aus den Daten Informationen, die zur weiteren Verarbeitung genutzt werden können. Ein Datenbanksystem umfasst die beiden Komponenten Datenbankmanagementsystem (DBMS) sowie die zu veraltenden Daten selbst.
Ein DBMS muss die vier Aufgaben [2] erfüllen.

- Atomarität
- Konsistenzerhaltung
- Isolation
- Dauerhaftigkeit

Neben den vielen neu auf den Markt erschienen Technologien wie Document Store oder Key-Value Store ist das Relationale Datenbankmodell immer noch am weit verbreitet. [2].

[1] In der Literatur oft auch als **D**aten**B**ank**S**ystemen (DBS) oder Informationssystem bezeichnet. [5, pp. 3-4]
[2] Bekannt als ACID-Prinzip [5, pp.105] umfasst es **A**tomicity, **C**onsistency, **I**solation und **D**urability.

6.2.2. Monolithische Systeme

Als Monolithisch wird ein logisches System bezeichnet, wenn es in sich geschlossen, ohne Abhängigkeiten zu anderen Systemen operiert. Alle zur Erfüllung der Aufgaben benötigten Ressourcen sind im System selbst enthalten. Es müssen also keine Ressourcen anderer Systeme alloziert werden und somit ist auch keine Kommunikation oder Vernetzung notwendig.
Das System selbst muss jedoch nicht notwendigerweise aus nur einem Rechenknoten bestehen, sondern darf auch als Cluster implementiert sein.

6.2.3. Verteilte Systeme

Man kann zwischen physisch und logisch verteilten Systemen unterscheiden. Weiter kann das System auf verschiedenen Abstraktionsstufen betrachtet werden. So sind je nach Betrachtungsvektor unterschiedliche Aspekte relevant und interessant. [8]

physisch verteilte Systeme

Rechnernetze und Cluster-Systeme werden typischerweise als physisch verteiltes System betrachtet. Die Kommunikation zwischen den einzelnen Rechenknoten erfolgt nachrichtenorientiert und ist somit asynchron ausgelegt. Jeder Rechenknoten verfügt über exklusive Speicherressourcen und einen eigenen Zeitgeber.
Durch die Implementation eines Systems über mehrere unabhängige physische Rechenknoten kann eine erhöhte Ausfallsicherheit und/oder ein Performance-Gewinn erreicht werden.

logisch verteilte Systeme

Falls innerhalb eines Rechenknoten echte Nebenläufigkeit[3] oder Modularität[4] erreicht wird, kann von einem logisch verteilten System gesprochen werden. Einzelne Rechenschritte und Aufgaben werden unabhängig voneinander auf der selben Hardware ausgeführt. Dies ermöglicht den flexiblen Austauschen[5] einzelner Aufgaben.

6.2.4. Verteilte Algorithmen

Verteilte Algorithmen sind Prozesse welche miteinander über Nachrichten (synchron oder asynchron) kommunizieren und so idealerweise ohne Zentrale Kontrolle eine Kooperation erreichen. [7]
Performance-Gewinn, bessere Skalierbarkeit und eine breitere Abdeckung der unterstützen von verschiedenen Hardware-Architekturen kann durch den Einsatz von verteilten Algorithmen erreicht werden.

[3] Von echter Nebenläufigkeit wird gesprochen, wenn verschiedene Prozesse zur selben Zeit ausgeführt werden. (Multiprozessor)

[4] Modularität beschreibt die Unabhängigkeit und Austauschbarkeit einzelner (Software-) Komponenten. (Auch Lose Kopplung gennant)

[5] Austauschbarkeit einzelner Programmteile wird durch die Einhaltung der Grundsätze von modularer Programmierung erreicht.

6.2.5. Verteilte Datenbanken

[Präsenzbibliothek ZHAW]

6.2.6. Replikation

Replikation vervielfacht ein sich möglicherweise mutierendes Objekt (Datei, Dateisystem, Datenbank usw.), um hohe Verfügbarkeit, hohe Performance, hohe Integrität oder eine beliebige Kombination davon zu erreichen. [1, p. 19]

Synchrone Replikation

Eine synchrone Replikation stellt sicher, dass zu jeder Zeit der gesamte Objektbestand auf allen Replikationsteilnehmern identisch ist.

Wird ein Objekt eines Replikationsteilnehmer mutiert, wird zum erfolgreichen Abschluss dieser Transaktion, von allen anderen Replikationsteilnehmern verlangt, dass sie diese Operation ebenfalls erfolgreich abschliessen.
Üblicherweise wird dies über ein Primary-Backup Verfahren realisiert. Andere Verfahren wie der 2-Phase-Commit und 3-Phase-Commit ermöglichen darüber hinaus auch das synchrone Editieren von Objekten auf allen Replikationsteilnehmern. [1, p. 23ff, 134ff]

Asynchrone Replikation

Eine asynchrone Replikation, stellt periodisch sicher, dass der gesamte Objektbestand auf allen Replikationsteilnehmern identisch ist. Mutationen können nur auf dem Master-Knoten durchgeführt werden. Einer oder mehrere Backup-Knoten übernehmen dann periodisch die Mutationen. Entgegen der synchronen Replikation müssen nicht alle Replikationsteilnehmer zu jedem Zeitpunkt verfügbar sein[6].

Merge Replikation

Die merge Replikation erlaubt das mutieren des Objekts auf einem beliebigen Replikationsteilnehmer.
Mutationen auf einem einzelnen Replikationsteilnehmer werden periodisch allen übrigen Replikationsteilnehmern mitgeteilt. Da ein Objekt zwischenzeitlich[7] auch auf anderen Teilnehmern mutiert worden sein kann, müssen während des Synchronisationsvorgang[8] eventuell auftretenden Konflikte aufgelöst werden.

[6] So kann der Backup-Knoten nur Nachts über verfügbar sein, damit die dazwischen liegende Verbindung Tags über nicht belastet wird.

[7] Zwischen der lokalen Mutation und der Publikation dieser an die übrigen Replikationsteilnehmer, liegt eine beliebige Latenz.

[8] Da die Replikation nicht notwendigerweise nur unidirektional, sondern im Falle von einem Multi-Master Setup auch bidirektional durchgeführt werden kann, wird hier von einer Synchronisation gesprochen.

6.2.7. Block-Chain

Die Block-Chain ist eine verteilte Datenbank die ohne Zentrale Kontrolle auskommt. Jede Transaktion wird kryptographisch gesichert, der Kette von Transaktionen hinzugefügt. So ist das entfernen oder ändern vorhergehender Einträge nicht mehr möglich[9]. Jeder Teilnehmer darf also alle Einträge lesen und neue Einträge hinzufügen. Da Einträge nur hinzugefügt werden und nie ein Eintrag geändert wird, kann eine Block-Chain immer ohne Synchronisationskonflikte repliziert werden. Konflikte können nur in den darüberlegenden logischen Schichten[10] auftreten. [6]

[9] Das ändern vorhergehender Einträge benötigt mehr Rechenzeit, als alle anderen Teilnehmer ab diesem Zeitpunkt zusammen aufgewendet haben.

[10] So prüft die Bitcoin-Implementation ob eine Transaktion (Überweisung eines Betrags) bereits schon einmal ausgeführt wurde, und verweigert gegebenenfalls eine erneute Ausführung.

7
Analyse

7.1. Datenanalyse

Daten können bezüglich ihrer Beschaffenheit, Geltungsbereich und Gültigkeitsdauer unterschieden werden. Dabei spricht man von der Klassifikation. Es werden nur die in den Daten enthaltenen Informationen dafür herangezogen. Die Form der Daten, also der Datentyp selbst ist für die Klassifikation unerheblich.

Die Datentypisierung unterscheidet zwischen numerischen (num), binären (bin), logischen (bool) und textuellen (text) Daten.

Eine Attributsgruppe, also eine logische Aufteilung der Informationen in mehrere Attribute ist eine zusammenhängende Informationseinheit.

7.1.1. Kontextbezogene Daten (Struktur)

Daten die nur in einem bestimmten Kontext einen signifikanten Informationsgehalt aufweisen, der ohne Kontext nicht greifbar ist, werden als kontextbezogene Daten bezeichnet.
(Kommentar auf einen Artikel)

7.1.2. Unabhängige Daten (Struktur)

Mit dem Begriff der unabhängigen Daten werden all jene Datensätze bezeichnet die in sich abgeschlossene Informationen beinhalten.
(Artikel)

7.1.3. Exklusive Daten (Art)

Exklusive Daten sind Daten und Datensätze die nur von einem Aktor bearbeitet werden dürfen.
(Wecker-Einstellungen)

7.1.4. Gemeinsame Daten (Art)

Daten die von mehreren Aktoren gleichzeitig gelesen und bearbeitet werden dürfen, werden als gemeinsame Daten bezeichnet.
(Firmen-Todo-Liste)

7.1.5. Dynamische Daten (Art)

Automatisch generierte oder sich sehr schnell verändernde Daten werden dynamische Daten genannt.
(Inhalt im Kühlschrank)

Dynamische Daten werden vom Benutzer nicht verändert, und müssen desshalb auch nicht synchronisiert werden.

7.1.6. Statische Daten (Art)

Daten die über einen grossen Zeitraum hinweg nicht an Gültigkeit verlieren werden statische Daten genannt.
(Koch-Rezept)

Statische Daten werden vom Benutzer nicht verändert und müssen desshalb auch nicht synchronisiert werden.

7.1.7. Temporäre Daten (Art)

Als Temporäre Daten können all jene Daten bezeichnet werden, die nur für einen sehr begrenzten Zeitraum gültig sind. Diese Daten sind auch immer exclusive Daten.
(Transaktions Logs)

7.2. Diskussion Synchronisationsproblem

- Bearbeitung gemeinsamer Daten, wer gewinnt? (letzter Veränderer, erster Synchronisierer, Berechtigungshierarchie)
- Bearbeiten von eigenen Daten auf unterschiedlichen Geräten
-

7.3. Diskussion bekannter Verfahren

7.4. Anforderungsanalyse

7.5. Vorgehensweise

7.5.1. Use-Cases

UC1: Lesen eines Elements (online)
UC2: Einfügen eines neuen Elements (online)
UC3: Ändern eines Elements (online)
UC4: Löschen eines Elements (online)
UC5: Lesen eines Elements (offline)
UC6: Einfügen eines neuen Elements (offline)
UC7: Ändern eines Elements (offline)
UC8: Löschen eines Elements (offline)

7.5.2. Anforderungen

7.5.3. Akzeptanzkriterien

7.5.4. Bewertung der Anforderungen

7.6. Risiken

Teil III.

Konzept und Implementierung

8
Konzept

Im Rahmen dieser Bachelorarbeit werden zwei grundsätzliche Umgangsmethodiken mit Synchronisationen bzw. Synchronisationskonflikten betrachtet.
Synchronisationen können so gestaltet werden, dass keine Synchronisationskonflikte auftreten, oder es können auftretende Konflikte gelöst werden.

8.1. Singlestate

Ein Single-State System lässt zu jedem Zeitpunkt t nur einen einzigen gültigen Zustand zu. Eingehende Nachrichten N enthalten sowohl die Änderungsfunktion als auch eine Referenz auf welchen Status S diese Mutation angewendet werden soll.

In Abbildung 8.1 sind die nacheinander eingehenden Nachrichten N_1 bis N_4 dargestellt. Nachricht N_2 sowie N_3 referenzieren auf den Status S_2. Die Anwendung der Änderungsfunktion von N_2 auf S_2 resultiert im gültigen Status S_3.
Die Anwenung der später eingegangene Nachricht N_3 auf S_2 führt zum ungültigen Status S_3'.

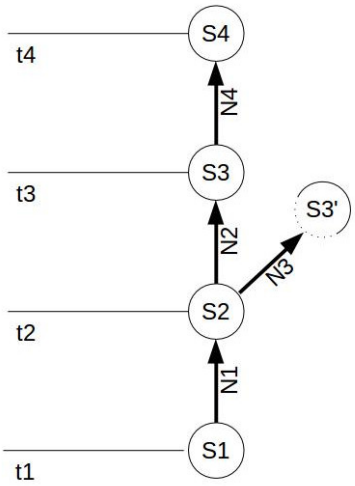

Abbildung 8.1.: Singlestate

Sobald eine Nachricht vom Server verarbeitet wurde die Transaktion abgeschlossen. Entweder ergab sich daraus ein neuer gültiger, oder aber ein ungültiger Status. In diesem Fall konnten die Mutationen nicht übernommen werden und wurden abgelehnt.

8.2. Multistate

Der Zustand des Systems wird durch eine Message-Queue repräsentiert.

1. Message gets sent
2. Message gets enqueued
3. If message conflicts with an other message -> next else -> end
4. message gets moved bevore conflicts
5. Fork is generated
6. It is decided with fork is the main fork (master)

Zu jedem Zeitpunkt sind beliebig viele gültige Stati erlaubt.
Es gibt zu jedem echten Zeitpunkt nur einen einzigen gültigen Status (main fork), bei Konflikten kann aber ein anderer Zweig nachträglich als Master definiert werden.

In Abbildung 8.2 sind die nacheinander erstellten Nachrichten N_1 bis N_5 und ihre Interaktion mit den Stati S_1 bis S_7 aufgeführt.
Die Nachrichten sind entsprechend ihrem Eingang beim Server geordnet.

Die echte Kausalität verhält sich jedoch wie folgt: $N_1 < N_2 < N_4 < N_5 < N_3$
Die Nachricht N_2 löst einen Konflikt mit der Nachricht N_4 aus. Alle übrigen Nachrichten sind konfliktfrei.

Mit M_1 wird das manuelle Zusammenführen zweier Stati bezeichnet.

Nach dem Eingang der Nachricht N_2 wird vom System der Status $S_{3.0}$ als aktuell gültiger Status für den Zeitpunkt t_3 geführt.
Sobald die Nachricht N_4 verarbeitet wurde, wird für den Zeitpunkt t_3 jedoch der Status S_4 als gültig gesetzt.

Zwischen t_3 und t_5 gehen die Nachrichten 3 und 5 ein. Beide Nachrichten werden auf den Status S_3 sowie auf die entsprechenden Stati S_4 und S_5 angewendet.

Zum Zeitpunkt t_5 existieren also der Status S_6 mit allen Mutationen ausser denen von N_2 und der Status $S_{3.1}$ mit allen Mutationen, ausser denen von N_4.

Entweder wird nun ein Teilbaum abgeschnitten oder wie gezeigt eine manuelle Zusammenführung M_1 durchgeführt

Jedem Benutzer/Session werden gegebenenfalls eigene Zweige zugewiesen. Das integrieren der Zweite in den Master-Zweig muss manuell vorgenommen werden.

Weiter werden alle neu eingehende Nachrichten für alle Zweige verarbeitet, ausser es entstünden dadurch weitere Konflikte.

Es können also immer alle Änderungen synchronisiert werden. Keine Mutationen gehen verloren. Die Konfliktauflösung kann nachträglich durchgeführt werden.

Wenn Nachrichten eingespielt werden, die Konflikte auflösen, kann ein anderer Zweig als Master markiert werden, dadurch können nachträglich andere Konflikte ausgelöst werden etc.

Z.B. das Resultat der Abfrage des Status gestern um 10:00 muss nicht dem Resultat der Abfrage von heute, wie der Status gestern um 10:00 war.

Jeder Status ist also Rückwirkend veränderbar. Entsprechende Systeme müssen dafür ausgelegt sein.

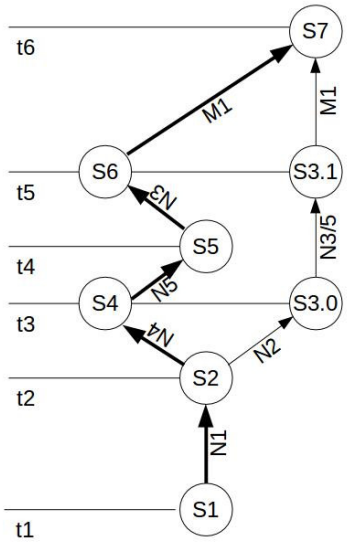

Abbildung 8.2.: Multistate

Die Konfliktauflösung funktioniert genau gleich wie bei den Singlestate Systemen. Wir dürfen jedoch sehr fiel mehr „greedy“ sein.

Statt das wahrscheinlichsten oder des erste eingehenden Attribut zu verwenden wird die Kausalität sichergestellt. So wird für jede Mutation ein „quasi“ Timestamp generiert (auch auf den mobilen Endgeräten) um so den Zeitpunkt der Synchronisierung zu egalisieren. Mutationen die echt zu erst durchgeführt wurden, gelten.

sehr greedy. Wir dürfen alles annehmen, und kümmern uns erst später um auftretende Fehler.

8.3. Konfliktvermeidung

Das Konzept der Konfliktvermeidung verhindert das Auftreten von möglichen Konflikten durch die Definition von Einschränkungen im Funktionsumfang der Datenbanktransaktionen. So sind Objekt aktualisierende Operationen nicht möglich und werden stattdessen, Client seitig, über hinzufügende Operationen ersetzt.

8.3.1. Update Transformation

Damit Mutationen für eines oder mehrere Attribute konfliktfrei synchronisiert werden können, wird die Änderung als neues Objekt der Datensammlung hinzugefügt.

Änderungen eines Attributes Ex werden als neues Attribut in einem neuen Objekt $Ix(Ex)$ erfasst. (Abbildung 8.3)

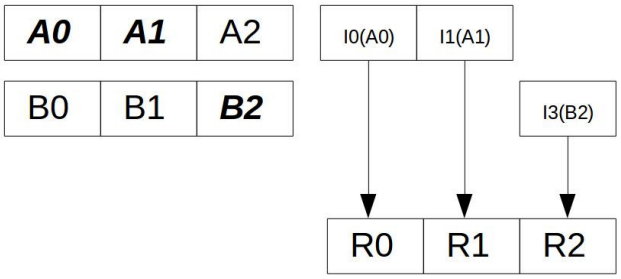

Abbildung 8.3.: Update Transformation

Solange die Einfüge-Funktion $Ix(Ex)$ eine Numerische Operation ($+$ oder -) ist, spielt es meist! Rolle in welcher Kausalität die Funktionen auf dem Server angewendet werden.

Kontextbezogene Daten

Kontextbezogene Daten können nur aktualisiert werden, wenn der Kontext sich nicht geändert hat.

Unabhängige Daten

Unabhängige Daten können ohne Abhängigkeit des Zustandes anderer Daten aktualisiert werden.
Zu beachten ist, dass nur numerische, boolsche und binäre Werte und nur die beiden Grundoperationen $+$ und - immer funktionieren.

Exklusive Daten

Für den Fall dass von 2 Geräten ein Update ausgeführt wird, müssen beide Versionen gespeichert werden, und der User muss entscheiden welche Version verwendet werden soll

Gemeinsame Daten

Im Falle von Text-Daten kann diese Art der Synchronisation nur für den tatsächlichen Add verwendet werden.

8.3.2. Wiederholbare Transaktion

Statt nur einer Überführung einer Einzelnen Mutation in ein Insert wird die gesamte Transaktion so gemacht.
Falls nun eine Leseaktion auf eine bereits mutiertes Objekt geschieht, welches noch nicht synchronisiert wurde, und aufgrund dieser Lesekation eine andere Mutation passiert, darf die zweite Mutation nur synchronisiert werden, falls die erste Mutation auch erfolgreich war.

Es werden applikatorische Inkonsistenzen vermieden. Zusätzlich müssen auf dem Client alle Lese- und Schreib-Operationen „ge-trackt“ werden.

Kontextbezogene Daten

können in einer Transaktion, nur aktualisiert werden, wenn der Kontext sich nicht geändert hat.

Unabhängige Daten

können ohne Abhängigkeit des Zustandes anderer Daten aktualisiert werden.
Zu beachten ist, dass nur numerische, boolsche und binäre Werte und nur die beiden Grundoperationen + und - immer funktionieren.

Exklusive Daten

Für den Fall dass von 2 Geräten ein Update ausgeführt wird, müssen beide Versionen gespeichert werden, und der User muss entscheiden welche Version verwendet werden soll

Gemeinsame Daten

Im Falle von Text-Daten kann diese Art der Synchronisation nur für den tatsächlichen Add verwendet werden.

8.4. Konfliktauflösung

Das Konzept der Konfliktauflösung beschäftigt sich mit der Auflösung von Konflikten, die im Rahmen der Synchronisation aufgetreten sind.
Da die Beschaffenheit und Struktur der Daten, bei dieser Problemstellung eine entscheidende Rolle einnehmen, ist im folgendn für jede Klassifikations-Gruppe ein geeigneter Konfliktauflösungs-Algorithmus aufgeführt.

8.4.1. Zusammenführung (Merge)

Einzelne Attribute oder Attributsgruppen innerhalb eines Objekts werden als eigenständige Objekte betrachtet. So kann ein Konflikt, der auftritt wenn zwei Objekte mit Mutationen in unterschiedlichen Attributsgruppen synchronisiert werden , aufgelöst werden, indem nur die jeweils mutierten Attributgrupen als synchronisationswürdig betrachtet werden.

Kontextbezogene Attribute sind in der selben Attributgruppe wie der Kontext.

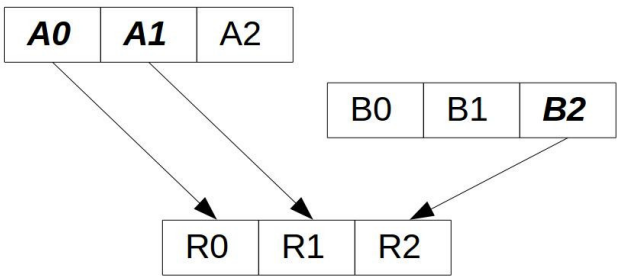

Abbildung 8.4.: Merge

Diese Operation ist unabhängig von Datentyp, Struktur und Art.
Der (automatisierte) Merge kann nur durchgeführt werden, wenn nur eine einzige Version eines Attributs existiert. -> nie konkurrierende Versionen entstehen.

Darüber hinaus versieht der Server jede Attribut-Version mit einer Versions-Nummer. So kann verhindert werden, dass ein Attribut mit einer niederen Versions-Nummer über ein neueres Attribut synchronisiert wird.

8.4.2. normalisierte Zusammenführung (Normalized Merge)

Wenn bei einer Synchronisierung mit zwei Objekten die selben Attribute mutiert wurden, kann im Falle von numerischen Attributen, das Objekt mit den geringsten Abweichungen vom Meridian über alle verfügbaren Datensätze verwendet werden. Es wird also das normalisierteste Attribut verwendet.
Bei Attributsgruppen wird die Gruppe mit der insgesamt geringsten Abweichung verwendet.
Es muss eine Abstandsfunktion für jedes Attribut oder jede Attributgruppe erstellt werden.

Es kann so die warscheinlichste Version verwendet werden. Wenn zu t_1 A_1 und zu t_2 A_2, also das selbe Attribut synchronisiert wird, wird es beide male angenommen. Beide Versionen

basieren auf der gleichen Ursprungsversion. A_2 besitzt aber die kleinere Abweichung und gewinnt deshalb.
Hätte es die grössere Abweichung, würde es nicht synchronisiert werden.

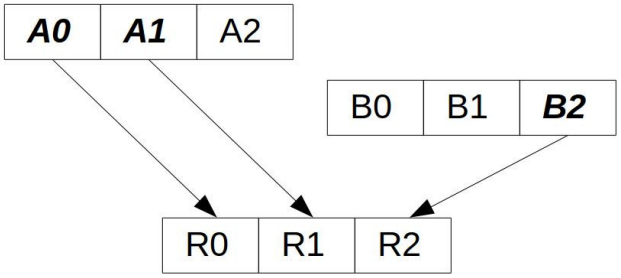

Abbildung 8.5.: Merge

8.5. Gesamtkonzept

Das Gesamtkonzept besteht in einer Zusammenführung aller bereits erwähnter Konzepte. Je nach zu synchronisierendem Datenbestand müssend entsprechend Passende Lösungsverfahren angewendet werden.

9
Design des Prototypen

In diesem Kapitel werden die aus dem Kapitel Konzept gewonnenen Erkenntnisse umgesetzt.

9.1. Design-Ansätze

Zur Lösung der Aufgabenstellung wurden drei Design-Ansätze erarbeitet. Diese werden folgend kurz erläutert.

9.1.1. Server zentrierte Architektur

Der Server führt alle Berechnungen o.ä. durch. Nur mit einer aktiven Verbindung zum Server können Manipulationen am Datenbestand durchgeführt werden.

9.1.2. Client zentrierte Architektur

Der Client trifft Entscheidungen und führt die Berechtigungsprüfung durch. Die Resultate werden dann dem Server übermittelt.

9.1.3. Client zentrierte, Server basierte Architektur

Der Client simuliert alle Manipulationen. Der Server entscheidet über das Resultat.

9.2. Entscheid

Anforderung UC 5-8 => {Client zentrierte, Server basierte Architektur, Message Oriented}

9.3. Design

Der Prototyp besteht aus 3 Bausteinen; Server, API und Client.

Abbildung 9.1.: Bausteinübersicht

Die Bausteine werden in den folgenden Kapitel erläutert.

9.3.1. Backend

Alle Daten müssen zur Aufbereitung in das Backend transferiert werden.
Im Backend wird zwischen Persistenz- und Logik-Schicht unterschieden.

Logik

Die Logik-Schicht nimmt alle Nachrichten entgegen und führt, sofern aufgetreten Konfliktauflösungen vor.
Die Kommunikation mit der API findet nur über Nachrichten statt.
Die Kommunikation mit der darunter liegenden Persistenz-Schicht findet über eine Asynchrone API statt.

S_LOGIC_SM_get
S_LOGIC_SM_create
S_LOGIC_SM_update
S_LOGIC_SM_delete

Persistenz

Die Persistenz soll Modell-Basiert sein.

9.3.2. API

Message Queuing & Message Passing - nothing else

Umsetzung mit REST-Like Verhalten (get,put,update,delete)

Client-Side
S_API_WEB_get
S_API_WEB_put
S_API_WEB_update
S_API_WEB_delete

Server-Side
S_API_WEB_send

9.3.3. Frontend

Der Client bietet keine Persistenz über einen Neustart hinweg.

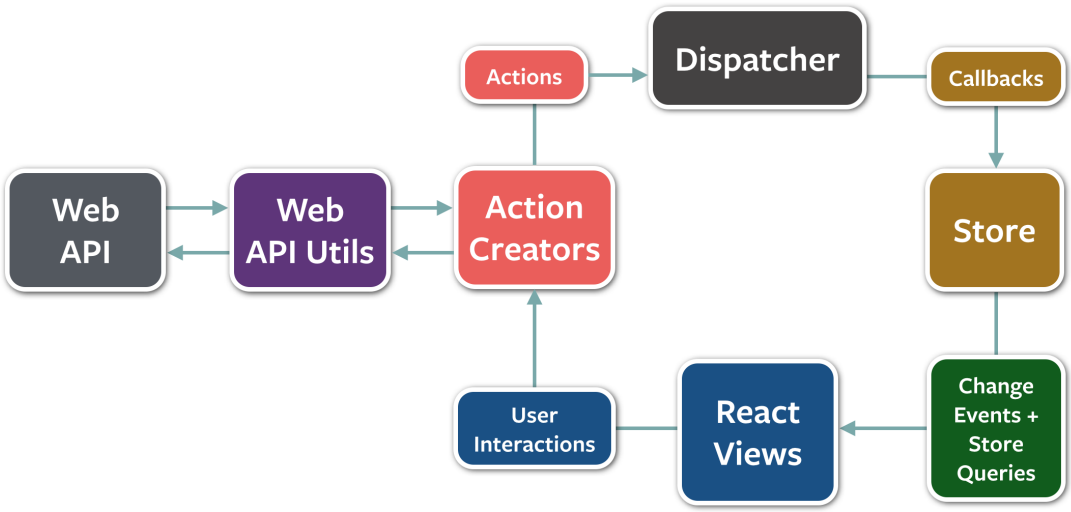

Abbildung 9.2.: Flux Diagramm

C_PRES_STORE_update
C_PRES_STORE_delete

9.3.4. Message Flow

Frontend <-> API <-> Backend

Frontend: ActionCreator -> Dispatcher -> Store -> API -> ActionCreator

API: Queue -> Transporter

Backend: API -> Logiclayer -> API

9.3.5. AMD Pattern

Asyncronous module definition (AMD) ist eine JavaScript API um Module zu definieren und diese zur Laufzeit zu laden. Dadurch können Javascript-lastige Webseiten beschleunigt werden, da Module erst geladen werden, wenn sie gebraucht werden. Weiter werden durch den Loader die Module gleichzeitig geladen, dadurch kann die Bandbreite voll ausgenutzt werden.
Da die Module durch die Vorgabe des Patterns in einzelnen Dateien abgelegt sind, wird eine Kapselung ähnlich wie bei Java erreicht. Das erleichtert die Fehlersuche und erhöht die Verständlichkeit des Programmes drastisch. Auch die Wiederverwendbarkeit der Module wird dadurch erhöht.
Da in jedem Modul die Abhängigkeiten definiert werden müssen, kann während dem Build-Prozess die Abhängigkeiten geprüft werden, um so die Verfügbarkeit aller benötigten Module sicher zu stellen.

9.4. Beispielapplikation

Gem. Aufgabenstellung soll der Prototyp anhand eines passenden Fallbeispiel die Funktionsfähigkeit Zeigen.

Die Beispielapplikation soll eine Ressourcenplan-Software sein. Folgendes soll möglich sein:

1. einen neuen Raum erfassen (Name, Grösse, Anzahl Sitze)
2. einen bestehenden Raum anpassen/löschen
3. einen Termin auf einem Raum Buchen (Name, Zeit&Datum, Kurzbeschreibung, Besucherliste, persönliche Notizen)
4. einen Bestehenden Termin anpassen/absagen

10 Prototyp

Dieses Kapitel adressiert die Implementation des Prototypen gemäss den Anroderungen aus Kapitel Analyse.

10.1. Umsetzung

10.2. Technologie Stack

Software	Beschreibung/Auswahlgrund
Grunt	Grunt ermöglicht es dem Benutzer vordefinierte Tasks von der Kommandozeile aus durchzuführen. So sind Build- und Test-Prozesse für alle Benutzer ohne detaillierte Kenntnisse durchführbar. Da Grunt eine sehr grosse Community besitzt und viele Plugins sowie hervorragende Dokumentationen verfügbar, wurde Grunt eingesetzt.
Karma	
CoffeeScript	
RequireJS	RequireJS ermöglicht die Implementierung des AMD Pattern.Dadurch können auch in JavaScript Code-Abhängigkeiten definiert werden. Zusammen mit r.js kann dies bereits zur Compilierzeit geprüft werden. Da weder Backbone noch Django über eine Depencency-Control für JavaScript verfügen, setze ich RequireJS ein.
ReactJS	
FluxifyJS	
SequelizeJS	
Express	
Socket.io	

10.3. Entwicklungsumgebung

Grunt + Karma = All you need

10.4. Entwicklung

Verwendung vom Socket io (Namespaces etc)
Tricks mit API & Message Routing, binding to io.on „message“ -> flux.doAction

Express Server:
Statisches Daten -> Frontend /
Socket.io -> /socket.io
Message-Bus -> Fluxify also in the backend

RequireJS Modules testable in the Browser :-D

Stores in the Frontend
Models in the Backend

10.5. Grafische Umsetzung Fallbeispiel

11 Testing

11.1. Unit-Testing

11.2. Integration-Testing

11.3. Test der Akzeptanzkriterien

11.4. Überprüfung der Aufgabenstellung

Teil IV.

Abschluss und Ausblick

12 Review

12.1. Testing

12.1.1. Integrationstest

12.1.2. Systemtest

12.2. Validation

12.2.1. Test der Akzeptanzkriterien

12.2.2. Überprüfung der Aufgabenstellung

13
Ausblick

14

Fazit und Schlusswort

A
Appendix

A.1. Glossar

A.2. Aufgabenstellung

A.2.1. Thema

Zeil der Arbeit ist es verschiedene Konfliktlösungsverfahren bei Multi-Master Datenbanksystemen zu untersuchen.

A.2.2. Ausgangslage

Mobile Applikationen (Ressourcen-Planung, Ausleihlisten, etc.) gleichen lokale Daten mit dem Server ab. Manchmal werden von mehreren Applikationen, gleichzeitig, dieselben Datensätze mutiert. Dies kann zu Konflikten führen. Welche Techniken und Lösungswege können angewendet werden, damit Konflikte gelöst werden können oder gar nicht erst auftreten?

A.2.3. Ziele der Arbeit

Das Ziel der Bachelorthesis besteht in der in der Konzeption und der Entwicklung eines lauffähigen Software-Prototypen, welcher mögliche Synchronisations- und Konfliktlösungsverfahren von Clientseitiger und Serverseitiger Datenbank demonstriert. Im Speziellen, soll gezeigt werden, welche Möglichkeiten der Synchronisation beim Einsatz von mobilen Datenbanken (Web-Anwendungen) bestehen, so dass die Clientseitige Datenbank auch ohne Verbindung zum Server mutiert und erst zu einem späteren Zeitpunkt synchronisiert werden kann, ohne dass Inkonsistenzen auftreten. Die Art und Funktionsweise des Software-Prototyp soll in einer geeigneten Form gewählt werden, so dass verschiedene Synchronisations- und Konfliktlösungsverfahren an ihm gezeigt werden können. Der Software-Prototyp soll nach denen, im Unterricht behandelten Vorgehensweisen des Test Driven Development (TDD) entwickelt werden.

A.2.4. Aufgabenstellung

A1 Recherche:

- Definition der Fachbegriffe
- Erarbeitung der technischen Grundlagen zur Synchronisation von Datenbanken und Datenspeichern

A2 Analyse:

- Analyse der Synchronisationsverfahren und deren Umgang mit Konflikten
- Analyse der Synchronisationsverfahren im Bereich der Web-Anwendungen
- Durchführen einer Anforderungsanalyse an die Software

A3 Konzept:

- Erstellen eines Konzepts der Software
- Erstellen eines Konzepts der Implementierung zweier ausgewählten Synchronisations-Verfahren

A4 Prototyp:

- Konzeption des Prototypen der die gestellten Anforderungen erfüllt
- Entwickeln des Software-Prototyps
- Implementation zweier ausgewählter Synchronisations- und Konfliktlösungsverfahren

A5 Review:

- Test des Prototyps und Protokollierung der Ergebnisse

A.2.5. Erwartete Resultate

R1 Recherche:

- Glossar mit Fachbegriffen
- Erläuterung der bereits bekannten Synchronisation- und Konfliktlösungs-Verfahren, sowie deren mögliches Einsatzgebiet

R2 Analyse:

- Dokumentation der Verfahren und deren Umgang mit Synchronisation-Konflikten (Betrachtet werden nur MySQL, MongoDB)
- Dokumentation der Verfahren zur Synchronisation im Bereich von Web-Anwendungen (Betrachtet werden nur die Frameworks Backbone.js und Meteor.js)
- Anforderungsanalyse der Software

R3 Konzept:

- Dokumentation des Konzepts der Software
- Dokumentation der Umsetzung der ausgewählten Synchronisations-Verfahren

R4 Prototyp:

- Dokumentation des Prototypen
- Implementation des Prototypen gemäss Konzept und Anforderungsanalyse
- Implementation zweier ausgewählter Synchronisations- und Konfliktlösungsverfahren

R5 Review:

- Protokoll der Tests des Software Prototypen

B
Verzeichnisse

B.1. Quellenverzeichnis

[1] Bernadette Charron-Bost. *Replication: Theory and Practice*. Hrsg. von Fernando Pedone und André Schiper. Lecture Notes in Computer Science. Berlin, Heidelberg: Springer Berlin Heidelberg, 2010. ISBN: 978-3-642-11294-2.

[2] *DB-Engines Ranking*. http://db-engines.com/de/ranking. [Online; accessed 19-March-2015]. 2015.

[3] Mike Elgan. *The hottest trend in mobile: going offline!* http://www.computerworld.com/article/2489829/mobile-wireless/the-hottest-trend-in-mobile--going-offline-.html. [Online, accessed 13-Januar-2015]. 2014.

[4] Andrew Lipsman. *Major Mobile Milestones in May: Apps Now Drive Half of All Time Spent on Digital*. http://www.comscore.com/Insights/Blog/Major-Mobile-Milestones-in-May-Apps-Now-Drive-Half-of-All-Time-Spent-on-Digital. [Online, accessed 13-Januar-2015]. 2014.

[5] Andreas Meier. *Relationale und postrelationale Datenbanken*. Bd. 7. eXamen.pressSpringerLink. Berlin, Heidelberg: Springer Berlin Heidelberg, 2010. ISBN: 978-3-642-05256-9.

[6] Satoshi Nakamoto. *Bitcoin: A Peer-to-Peer Electronic Cash System*. https://bitcoin.org/bitcoin.pdf. 2009.

[7] *Verteilte Algorithmen*. https://www.vs.inf.ethz.ch/edu/HS2014/VA/Vorl.vert.algo12-All.pdf. [Online, accessed 19-March-2015]. 2014.

[8] *Verteilte Systeme*. https://www.vs.inf.ethz.ch/edu/HS2014/VS/slides/VS-Vorl14-all.pdf. [Online, accessed 19-March-2015]. 2014.

B.2. Tabellenverzeichnis

B.3. Abbildungsverzeichnis