Masterarbeit

Ein System zur partiellen Synchronisation von Wissensbasen für dezentrale soziale Netzwerke

von Jens Grundmann
26. September 2015

Hochschule für Technik und Wirtschaft Berlin
Fachbereich Wirtschaftswissenschaften II
Studiengang Angewandte Informatik

Erstgutachter/in Prof. Vorname Name
Zweitgutachter/in Vorname Name

Hochschule für Technik
und Wirtschaft Berlin
University of Applied Sciences

Inhaltsverzeichnis

1 Einleitung **2**

2 Grundlagen **3**
2.1 Funktionale und nicht funktionale Anforderungen 3
2.2 Shark Framework 3
2.2.1 Context Space 4
2.2.2 Knowledge Base 5
2.2.3 Knowledge Port 5
2.2.4 SyncKB 6
2.3 Datenstrukturen zur Darstellung von Beziehungen 8
2.3.1 Verkette Listen 8
2.3.2 Bäume 9

3 Konzeption **10**
3.1 Ein erweiterter Kontext 10

4 Quellenverzeichnis **11**

5 Abbildungsverzeichnis **12**

1 Einleitung

Soziale Netzwerke erfreuen sich immer mehr Beliebtheit in den letzten Jahren. Sei es Facebook oder schlicht das Forum zum Online Game, dass man gerade spielt. Der Mensch möchte sich austauschen. Allerdings spätestens seit den Enthüllungen Edward Snowdens gegen Ende Mai 2013 stellt sich hier die Frage der Sicherheit der persönlichen Daten. Die meisten sozialen Netzwerke basieren auf einer Client-Server Architektur. Dies beutetet, dass alle Daten auf einem entfernten Server gespeichert sind. Der Nutzer hat somit keine Kontrolle bzw. nur die beschränkte Kontrolle, welche der Betreiber des sozialen Netzwerkes anbietet, wie mit seinen Daten umgegangen wird.

Dies ist Motivation für ein Umdenken. Anstatt die Daten an zentraler Stelle zu speichern verbleiben sie auf den lokalen Systemen der Benutzer. Das Client-Server Modell wird durch ein Peer to Peer Model ersetzt. Daten werden nur mit den Personen ausgetauscht, für die sie bestimmt sind. Hier ist eine Methode benötige, welche die Daten der lokalen Systeme mit einander synchronisiert.

Ziel dieser Arbeit ist es eine Softwarekomponente zu entwickeln, die eine partiellen Synchronisation von Wissensbasen ermöglicht. Eine Wissensbasis ist dabei nichts anderes als ein eine Menge an Daten, die in einer bestimmten Stuckatur vorliegen bzw. durch eine abstrakte Darstellung beschreibbar sind. Mittels dieser abstrakte Darstellung soll die Implementierung von Chats, Foren bis hin zu Source Code Management Systemen auf einer Peer to Peer Basis vereinfacht werden. Als Grundlage dient hierzu das Shark Framework [7] von Prof. Dr. Thomas Schwotzer und die darin enthalte SyncKB Klassensammlung. Diese ermöglicht bereits eine Synchronisation aller Daten in einer Wissensbasis. Diese soll nun dahingehen ausgebaut werden, dass Teile der Wissensbasis beschrieben werden können und nur diese beschrieben Teile synchronisiert werden.

Zuerst muss mit eine Möglichkeit gefunden werden, wie ein Teilbereich der Wissensbasis beschrieben werden kann. Diese müssen dann zwischen den einzelnen Peers kommunizierbar und auf den lokalen Systemen der Peers persistierbar sein. Die durch diese Beschreibungen extrahierten Daten werden synchronisiert, wobei der Peer auf diese Aktion regieren kann um sie beispielsweise in einer grafischen Oberfläche auszugeben. Hierbei ist darauf zu achten, dass die Beschreibung so abstrakt gewählt wird, dass sie auf möglichst viele Fälle, wie die erwähnte Möglichkeit der Implementierung von Chats oder Foren, anwendbar ist.
à Als Beweis der Funktionalität der Softwarekomponente wird schließlich eine Chat mit grafisch Oberfläche geschrieben, in dem einzelne Peers sich miteinander unterhalten können.

Die Arbeit wird zuerst die Grundlagen, wie das Shark Framework [7], auf denen die Implementierung beruht, erläutern. Danach wird das Konzept der Softwarekomponente erstellt. Es werden verschieden Möglichkeit diskutiert, um die im letzten Absatz beschriebenen Anforderungen umzusetzen. Im Anschluss wird die eigentliche Implementierung vorgestellt und auf diese eingegangen. Nachfolgend werden die Tests und Methoden nur Qualitätssicherung gezeigt und erklärt. Zuletzt wird ein Fazit gezogen sowie ein Ausblick auf mögliche Verbesserungen oder Erweiterungen.

2 Grundlagen

Das folgende Kapitel widmet sich den Grundlagen, auf denen die Arbeit beruht. Zuerst werden die Funktionale und nicht funktionale Anforderungen festgelegt, da diese die Basis der zu entwickelnden Softwarekomponente darstellen. Anschließen wird auf das Shark Framework eingegangen, auf dem diese Arbeit aufbaut.

2.1 Funktionale und nicht funktionale Anforderungen

Im Folgendem werden die funktionalen und nicht funktionale Anforderungen besprochen. Diese beschreiben welche Features die Softwarekomponente bereitstellen soll, sowie wichtige Aspekte der Qualitätssicherung.

Funktionale Anforderungen

- **Beschreibbarkeit:** Es ist möglich einen Raum von Daten zu beschreiben und diesen von einem anderen Raum von Daten abzugrenzen.
- **Abhängigkeiten:** Es ist möglich Abhängigkeiten zwischen Räumen zu definieren. So soll beispielsweise der Raum Java-Chat ein Kind des Programmiersprachen-Chat Raumes sein können.
- **Persistenz:** Es soll möglich sein die Beschreibung der Räume von Daten persistent zu speichern. Die gespeicherten Räume bleiben somit erhalten und und können zu späterem Zeitpunkt neu geladen werden.
- **Synchronisation:** Es ist möglich die Räume von Daten und ihre Abhängigkeiten mit Peers in einem Peer to Peer Netzwerk zu synchronisieren. Ziel ist es, dass die Räume nach der Synchronisation identisch von Aufbau und Inhalt sind.
- **Änderbarkeit:** Es muss möglich sein eine Beschreibung eines Raumes zu ändern inklusive ihrer Abhängigkeiten. Der Raum selbst soll dabei bestehe bleiben und ohne weiteren Aufwand auffindbar wie zuvor. Das heißt, er muss genauso aus der seiner persistenten Form geladen werde können wie zuvor.
- **Änderung kommunizieren:** Änderungen müssen kommuniziert werden können, sodass sich andere Peers synchronisieren.

Nicht funktionale Anforderungen

- **Build-Management:** Die Softwarekomponente ist mittels eines zu bestimmenden Build Tools so eingerichtet, dass das Aufsetzen der Entwicklungsumgebung für andere Entwickler schnell und einfach zu erledigen ist. Mögliche Synergien des gewählten Tools mit anderen Systemen zur Softwareentwicklung und Qualitätssicherung, zum Beispiel Jenkins [2], sind wünschenswert.
- **Testbarkeit:** Die zu Softwarekomponente ist modular so aufgebaut, dass sie durch Modultest testbar ist.
- **Modultest:** Es existieren bereit eine Reihe von Modultest, welche die grundlegende Funktionalität der Softwarekomponente sicherstellen.

2.2 Shark Framework

In diesem Unterkapitel wird auf die grundlegenden Features des Shark Framework [7] eingegangen, die für die Arbeit benötigt werden. Das gesamte Framework wird nicht erklärt. Weiterführende Informationen sind im Developer Guide [6] zu finden.

2.2.1 Context Space

Die wichtigste Grundlage ist die Context Space, der hier vereinfacht Kontext genannt werden soll. Hierbei handelt es sich um eine Datenstruktur. Abbildung 1 zeigt ein vereinfachtes Modell dieser Struktur.

Abbildung 1: Shark Context Space Modell

Die einzelnen Elemente haben dabei folgende Bedeutung:

- **Topic:** Dies ist das Thema eines Kontextes. Es ist ist eine Beschreibung was der Kontext bedeutet.
- **Orginator:** Der Autor der des beschrieben Kontextes. Die kann entweder der Ersteller der sein oder Autor der beiliegen Informationen. Bei einem wissenschaftlichen Artikel wäre dies der Autor des Titels.
- **Peers:** Peers kann unterschiedlich interpretiert werden. Zum einen kann es der Ersteller des Kontextes sein, der sich vom Autor der beiliegenden Informationen unterschieden kann. Zum anderen kann es der Besitzer des eines Kontextes sein, wobei Besitzer hier je nach beiliegendem Fall anders interpretiert werden kann. Im Allgemeinen kommt die Interpretation dieser auf den vorliegenden Fall an.
- **Remote Peers:** Dies sind die Peers an welche der Kontext gesendet werden soll.

- **Times:** Eine Zeitinformation die je nach vorliegendem Fall anders Interpretiert weiden kann. Diese Dimension biete die Möglichkeit ein Zeitintervall anzugeben.
- **Locations:** Gibt einen Ort an. Diese Information kann je nach vorliegendem Fall anders interpretiert werden.
- **Direction:** Die Richtung der Kommunikation des Kontextes. Es ist möglich nur Informationen über einen Kontext zu empfangen, sie nur zu senden, zu lesen und senden und keine der genannten Aktionen durchzuführen.

Zu beachte ist, dass wenn von einem Kontext gesprochen wird, dann handelt es sich hierbei um eine Beschreibung einer Menge von Kontextpunkten. Diese Punkte haben den gleichen Aufbau wie in Abbildung 1 gezeigt. Der wesentliche Unterschied ist, dass ein Kontext eine Vielzahl von Inhalten, ausgenommen der Direction, besitzen kann. Wehrendessen besitzt der Kontextpunkt nur je genau einen Inhalt. Beispielhaft beschriebt der Kontextpunkt nur das Thema Java, während der Kontext Java und C++ beinhaltet. Die Dimensionen selbst werde als Kontextkoordinaten bezeichnet. Der Kontextpunkt vereint Koordinaten und Informationen. Der Kontext wiederum ist eine Möglichkeit eine menge an Kontextpunkten aus einer zugrundeliegenden Wissensbasis zu extrahieren.

2.2.2 Knowledge Base

Die Knowledge Base, zu deutsch Wissensbasis, ist Eine Sammlung von Wissen. Wissen ist dabei eine Menge an Informationen, die anhand von Kontextkoordinaten beschrieben sind. Die Vereinigung von Kontextkoordinaten und Informationen bildeten einen Kontextpunkt.

Die Wissensbasis bietet die Möglichkeit Peers und Themen zu speichern, sowie die Themen in einer Taxonom einzuordnen. Der für die zu entwickelnde Softwarekomponente wichtige Punkt ist aber die Möglichkeit genannte Kontextpunkt in ihr zu speichern und anhand eines Kontextes zu extrahieren. Die Wissensbasis selbst kann hierbei ähnlich einer Datenbank angesehen werden. Ziel ist es Teilbereiche, sprich eine bestimmte Menge an Daten dieser, mit anderen Wissensbasen zu synchronisieren. Zu diesem Zweck gebt es bereits eine Implementierung, SyncKB genannt, auf der aufgebaut wird.

Zusätzlich zu den genannten Eigenschaften gibt es die Properties an der Wissensbasis zu speichern. Dies sind Name-Wert-Paare, wobei sowohl Name als auch Wert eine Zeichenkette ist.

2.2.3 Knowledge Port

Das wichtigste Objekt zur Kommunikation im Peer to Peer Netzwerk vom Shark Framework ist der KnowledgePort. Hierbei handelt es um eine abstrakte Klasse, welche die Implementierung von *doExpose(SharkCS interest, KEPConnection kepConnection)* und *doInsert(Knowledge knowledge, KEPConnection kepConnection)* erfordert.

Die doExpose-Methode erhält ein Interesse in der Form eines SharkCS. Wenn ein Interesse versandt wurde, so wird sie als Erstes aufgerufen. Ihre generelle Aufgabe ist zu ermitteln, ob das Interesse für, das erhalten wurde, für den KnowledgePort von Relevanz ist.

Die doInsert-Methode hingegen erhält ein Knowledge Objekt, welches echte Daten enthält. Es wird in der Regel gegen Ende der Kommunikation auf gerufen und soll die Daten in die Wissensbasis einfügen.

Beide Methoden erhalten ein KEPConnection Objekt mit dem sie Informationen über den Sender erhalten können. Des Weiteren kann mithilfe dieses Objektes Daten an die doExpose und doInsert-Methode anderer Peer gesendet werden. Dies erlaubt einen mehrfachen Datentausch. Ein Beispiel hierzu ist der Synchronisationsprozess der SychKB, der im Nächten Abschnitt beschrieben wird und in Abbildung 2 skizziert ist.

Zu beachten ist, dass die Aussagen hier dem Standardfall entsprechen, wonach der Knowledge Port entworfen wurde. Die tatsächliche Implementierung und damit der Umgang mit dem SharkCS Objekt und dem Knowledge kann von jedem Entwickler selbst bestimmt werden.Einzig vorgeschrieben ist, dass zuerst ein Interesse versandt wird und Daten in der Form eines Knowledge Objektes gesendet werden.

2.2.4 SyncKB

Leider gibt es, abgesehen der Javadoc Dokumentation, keine genaue Beschreibung der Funktionsweise der SyncKB. Daher soll hier auf die generelle Funktionsweise dieser eingegangen werden. Weitere Informationen können den Quellcode selbst entnommen werden, der im Github vom SharkFW zu finden ist. [1]

Die SyncKB enthält den Hauptteil ihrer Logik im SyncKP. Die eigentliche Synchronisation findet während des Kommunikationsprozesses statt und ist keine Eigenschaft der Wissensbasis selbst. Die einzelnen Elemente der SyncKB basiert hauptsächlich auf dem Entwurfsmuster Wrapper. So wurden den einzelnen Element zwei zusätzliche Informationen mitgeben:

- **Version:** Die Aktuelle Version des Elements. Ein neu erstelltes Element startet bei 1 und bei jeder Änderung wird die Version um eins erhöht.
- **Zeitstempel:** Ein Zeitstempel der angibt, wann die letzte Änderung an dem Element vorgenommen wurde.

Wie erwähnt befindet sich der Großteil der Logik im SyncKP. Abbildung 2 zeigt die ablaufende Kommunikation zwischen zwei Peers, die hier vereinfacht Alice und Bob genannt werden sollen.

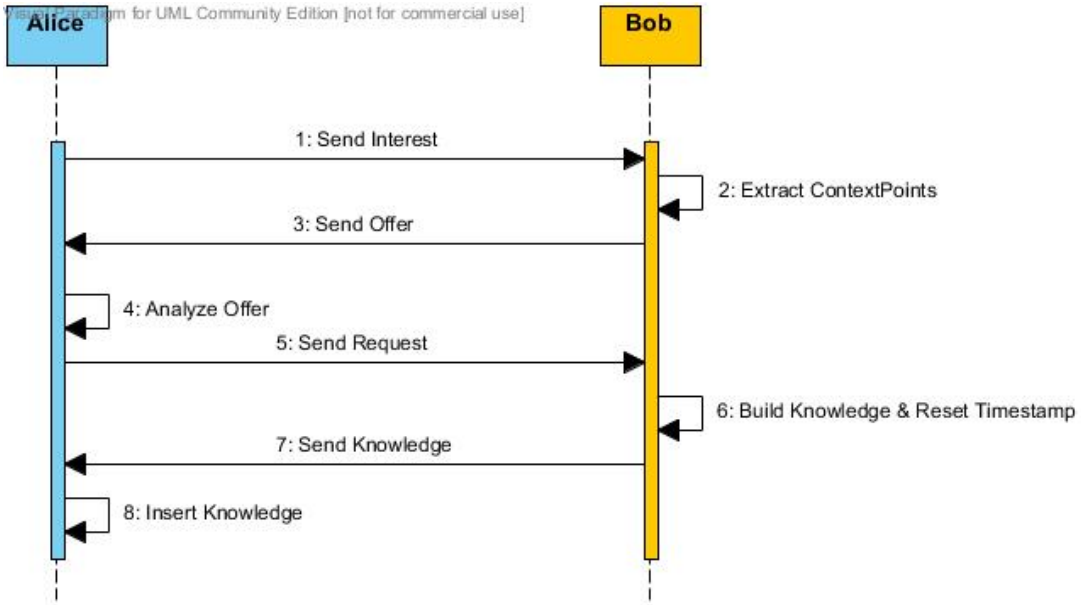

Abbildung 2: Kommunikation der SyncKB

1. **Interesse senden:** Der Prozess beginnt damit, dass Alice ihr Interesse zur Synchronisation verkündet. Das hierbei gesendete Interesse ist ein künstliches Interesse. Es wird vom SyncKP intern verwendet und dient lediglich als Datenhalter für diesen.
2. **Kontextpunkte extrahieren:** Nachdem das Interesse an der Synchronisation Bob erreicht hat stellt dieser ein Angebot für Alice zusammen. Im linken Teil von Abbildung 3 ist der Algorithmus dazu skizziert. Hierbei wird ausgenutzt, dass an jedem SyncContextPoint ein Zeitstempel der letzten Änderung gespeichert ist. Dieser wird mit dem Zeitpunkt des letzten Treffens mit dem Peer, das die Synchronisation anfordert, verglichen. Dazu ist an der Wissensbasis, per Property, eine Liste von Name-Wert-Paare gespeichert, welche

den Zeitpunkt des letzte Treffen mit einem Peer enthält. Genauer ist ein Peer jeweils einem Zeitstempel zugeordnet. Alle Kontextpunkte, deren letzte Änderung neuer ist als das letzte Treffe mit einem spezifischen Peer, hier Alice, werden angeboten.

3. **Angebot senden:** Die extrahierten Kontextpunkte werden per Property am künstlichen Interesse Alice als Angebot zugewendet. Zu beachten ist dabei, dass nur die Daten eines Kontextpunktes versendet werden. Eventuelle Informationen, die diesem zugeordnet sind, werden nicht versandt.

4. **Angebot analysieren:** Alice analysiert das Angebot, dass sie von Bob erhalten hat. Der Algorithmus ist ähnlich dem späteren Einfügen von der Kontextpunkte und im rechten Teil von Abbildung 3 skizziert. Alice wird alle Kontextpunkte von Bob anfragen, die sie nicht besitzt oder die bei Bob eine höhere Versionsnummer haben als bei ihr selbst.

5. **Anfrage senden:** Die anzufragenden Kontextpunkte werden abermals am künstlichen Interesse per Property gespeichert und Bob zugesandt.

6. **Wissen bauen und Zeitstempel zurücksetzen:** Die angefragten Kontextpunkte werden nun aus Bobs Wissensbasis extrahiert und in einem Knowledge Objekt gespeichert. Zusätzlich werden alle Themen und Peers, anhand der beim Erstellen des SyncKP übergebenen FragmentationParameter, dem Knowledge Objekt übergeben.

7. **Wissen senden:** Das Knowledge Objekt wird nun an Alice gesendet.

8. **Wissen in Wissensbasis einfügen:** Alice überprüft das die Kontextpunkte anschließend noch einmal anhand das im rechten Teil von Abbildung 3 skizzierten Algorithmus und fügt diese dann in ihre Wissensbasis ein.

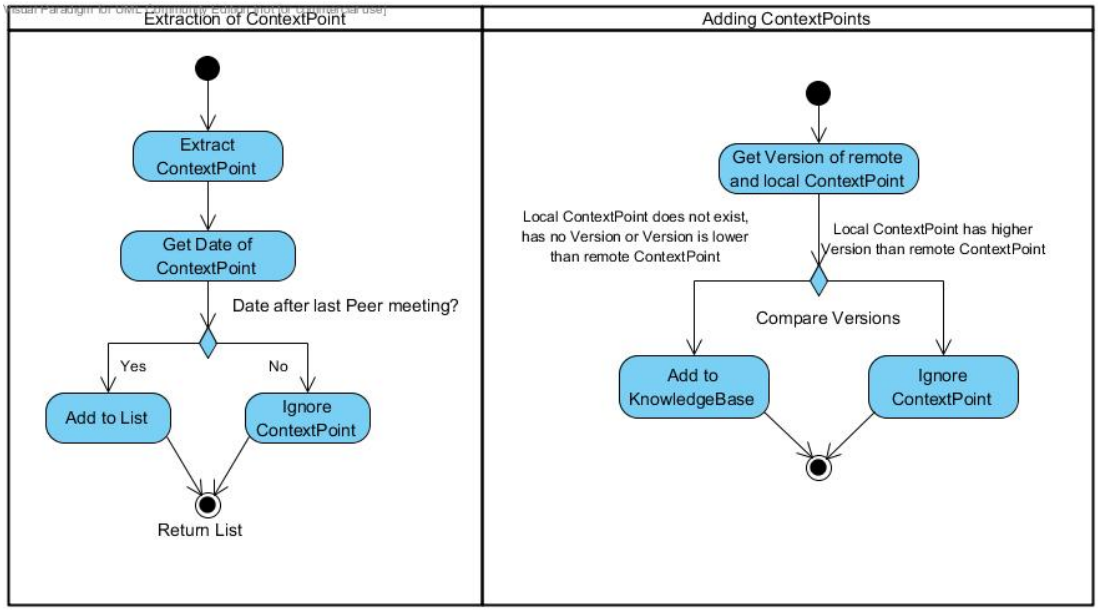

Abbildung 3: Algorithmus zum Extrahieren und Einfügen von Wissen der SyncKB

Das System von Angebot und Anfrage wird verwendet, um das Datenvolumen während der Kommunikation möglichst gering zu halten. Wie erwähnt enthalten Angebot und Anfrage nur die nötigen Informationen zu den Kontextpunkten, wie Koordinaten, Zeitstempel und Version, und zusätzlich mit dem Punkt verknüpften Informationen. Erst gegen Ende des Synchronisationsalgorithmus wird ein Objekt mit den vollständiges Daten erstellt und versandt.

2.3 Datenstrukturen zur Darstellung von Beziehungen

Wie in Abschnitt 2.1 beschrieben soll es möglich sein Abhängigkeiten zwischen Räumen aufzustellen. Diese Abhängigkeiten stellen eine Beziehung zwischen Daten da. Daher werden in diesem Abschnitt Datenstrukturen besprochen, die eine solche Beziehung darstellen können. Dabei wird darauf eingegangen, wie sie die Daten und Beziehungen untereinander dargestellt sind. Weitergehende Erklärungen, wie mögliche Operationen, werden nur durch Links zu entsprechender Literatur gegeben.

2.3.1 Verkette Listen

Verkette Listen (beschrieben in [5], Kapitel 4) sind Listen, wo jedes Element Referenzen auf weitere Mitglieder der Liste einhält. Hier sollen die Einfach-Verketteten-Listen und die Zweifach-Verketten-Listen betrachten werden.

Einfach-Verkettete-Listen

Einfach-Verkettete-Listen besitzen eine Referenz auf ihren Nachfolger. Die Beziehung der einzelnen Elemente ist hierbei, dass jedes Element seinen Nachfolger kennt, nicht aber seinen Vorgänger. Die Beziehungen können also nur vorwärts verfolgt werden, das heißt von einem Element Zum nachfolgenden, nicht aber von einem Element zum Vorherigen. Abbildung 4 skizziert dieses und zeigt mögliche Operation an einer verketteten Liste. weitere Informationen können der Erklärungen unter [3] entnommen werden.

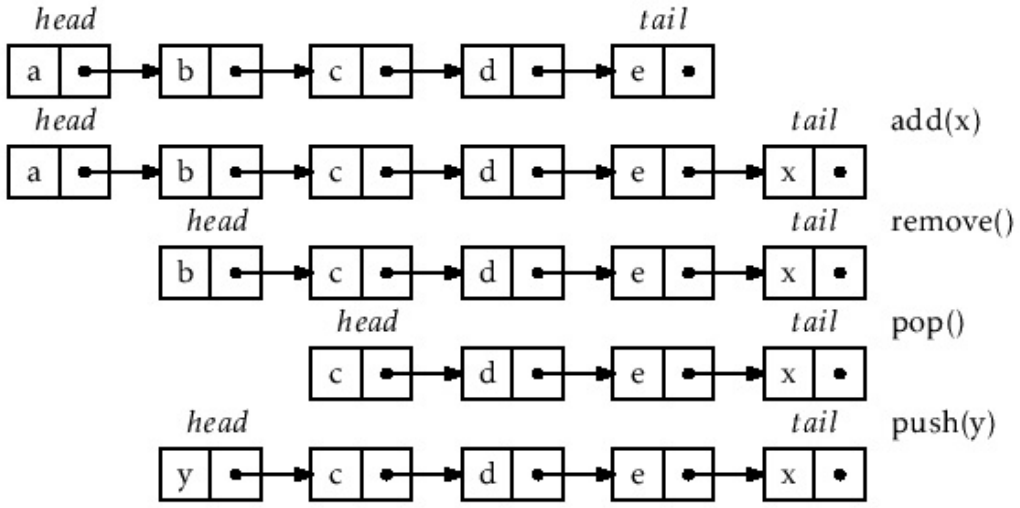

Abbildung 4: Aufbau und Operation einer einfach verketteten Liste. Quelle: [3]

Zweifach-Verkette-Listen

Zweifach-Verkette-Listen besitzen, gegenüber Einfach-Verkettete-Listen, eine Referenz auf Vorgänger und Nachfolger. Die Beziehung der einzelnen Elemente ist also, dass jedes Element seinen Vorgänger und Nachfolger kennt. Somit kann die Liste vorwärts, von Element zum nachfolgenden Element, als auch rückwärts, vom Element zum vorhergehenden Element verfolgt werden. Abbildung 6 skizziert dieses. Weitere Informationen können den Erklörungen unter [4] entnommen werden.

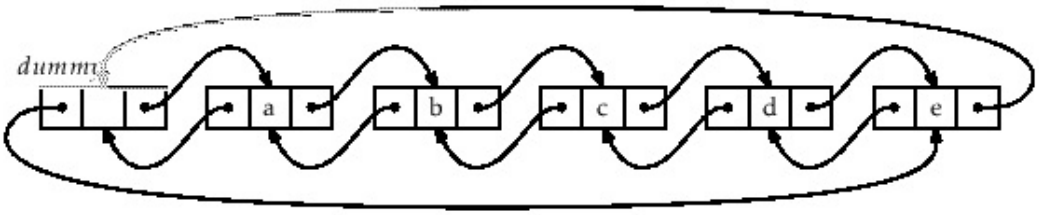

Abbildung 5: Aufbau einer zweifach verketteten Liste. Quelle: [4]

2.3.2 Bäume

Bäume ([5], Kapitel 4) bestehen aus einem Wurzelknoten und weiteren Knoten, die von diesem ausgehen. Hierbei besteht eine Vater-Kind Beziehung zwischen diesen. Die Wurzel besondere Eigenschaft der ist, dass sie keinen Vater besitzt. Knoten, die keine Kinder besitzen, werden Blätter genannt. Ein Element kann hier also beliebig vielen Unterelementen, seinen Kindern, in Beziehung stehen. Hingegen kann ein Element von nur einem anderen Element abstammen. Abbildung 6 skizziert ein Beispiel dieser Datenstruktur.

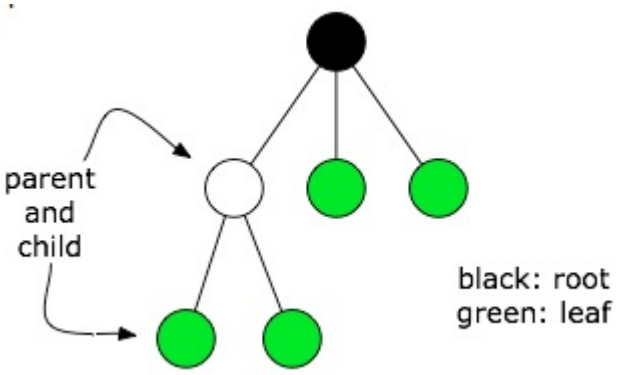

Abbildung 6: Aufbau eines Baums. Quelle: [8]

2.3.3 Taxonomy

3 Konzeption

3.1 Ein erweiterter Kontext

Die Context Spache aus dem Kapitel 2.2.1 ist für unsere Anforderungen nicht ausreichend.

4 Quellenverzeichnis

Bücher

[5] Ellis Horowitz und Sartaj Sahni. *Fundamentals of Data Structures*. Computer Science Press, 1976. ISBN: 091489420X.

Handbücher

[6] Prof. Dr. Thomas Schwotzer. *Building Semantic P2P Applications with Shark*. URL: `http://www.sharksystem.net/sharkDeveloperGuide/Shark2.0_DevelopersGuide.pdf` (besucht am 18. 09. 2015).

Webseiten

[1] *Github SharkFW - SyncKB*. URL: `https://github.com/SharedKnowledge/SharkFW/tree/master/src/java/core/net/sharkfw/knowledgeBase/sync` (besucht am 21. 09. 2015).

[2] *Jenkins*. URL: `http://jenkins-ci.org/` (besucht am 15. 09. 2015).

[3] *opendatastructures.org - 3.1 SLList: A Singly-Linked List*. URL: `http://opendatastructures.org/versions/edition-0.1g/ods-python/3_1_SLList_Singly_Linked_Li.html` (besucht am 26. 09. 2015).

[4] *opendatastructures.org - 3.2 DLList: A Doubly-Linked List*. URL: `http://opendatastructures.org/versions/edition-0.1g/ods-python/3_2_DLList_Doubly_Linked_Li.html` (besucht am 26. 09. 2015).

[7] Prof. Dr. Thomas Schwotzer. *Shark framework*. URL: `http://www.sharksystem.net/` (besucht am 31. 08. 2015).

[8] Paul E. Black und. *Dictionary of Algorithms and Data Structures*. URL: `http://xlinux.nist.gov/dads//HTML/tree.html` (besucht am 26. 09. 2015).

5 Abbildungsverzeichnis

1 Shark Context Space Modell 4
2 Kommunikation der SyncKB 6
3 Algorithmus zum Extrahieren und Einfügen von Wissen der SyncKB 7
4 Aufbau und Operation einer einfach verketteten Liste. Quelle: [3] 8
5 Aufbau einer zweifach verketteten Liste. Quelle: [4] 9
6 Aufbau eines Baums. Quelle: [8] 9