Distributed Discussions in Online Social Networks

Masterarbeit

Florian Müller
Betreuer: Prof. Dr. Max Mühlhäuser
Verantwortlicher Mitarbeiter: Dipl.-Inform. Kai Höver
Darmstadt, September 2013

TECHNISCHE
UNIVERSITÄT
DARMSTADT

Fachbereich Informatik
Telekooperation
Prof. Dr. Max Mühlhäuser

Ehrenwörtliche Erklärung

Hiermit versichere ich, die vorliegende Masterarbeit ohne Hilfe Dritter und nur mit den angegebenen Quellen und Hilfsmitteln angefertigt zu haben. Alle Stellen, die aus den Quellen entnommen wurden, sind als solche kenntlich gemacht worden. Diese Arbeit hat in dieser oder ähnlicher Form noch keiner Prüfungsbehörde vorgelegen.

Darmstadt, September 2013

_____

(Florian Müller)

Zusammenfassung

Inhalt...

Inhaltsverzeichnis

Abbildungsverzeichnis

Tabellenverzeichnis

Liste der noch zu erledigenden Punkte

1 Einleitung

- Motivation/Problemstellunng
- Anforderungen: was soll entwickelt werden
- Übersicht über alle Kapitel

Anfang

Jedem ist es heutzutage möglich eigene Inhalte ohne großen Aufwand ins Internet zu stellen und anderen an seinen Wissen teilhaben zu lassen. Dadurch was es noch nie so einfach an Wissen Dritter zu kommen wie heute und dieses Wissen in seinen eigenen Lernprozess mit einfließen zu lassen.

Soziale Netzwerke erleben seit den letzten Jahren einen großen Boom. Sie ermöglichen es mit seinen Freunden in Kontakt zu bleiben auch über zeitliche und räumliche Hürden hinweg. Diese sozialen Netzwerke erlauben es auch ohne physische Präsenz sich untereinander zu organisieren. Qiyun Wang et. al. [10] zeigten, dass zum Beispiel Facebook[1] sich hervorragend für den Einsatz als Lernplattform beziehungsweise Learning Management System (LMS) eignet. Gerade die Organisation der Lerngruppe als auch die Benachrichtigung über Ereignisse funktionierte reibungslos. Jedoch uneingeschränkt konnte Facebook als LMS nicht empfohlen werden. Bemängelt wurden unter anderem die aufwändige Integration von Lernmaterialien „tutor noticed that it was quite troublesome to add teaching materials“[10, S. 435]. Aber auch da es nicht möglich war Diskussionen in einzelne Themen zu unterteilen sondern alle Beiträge nur chronologisch angeordnet sind wurde bemängelt.

Für die meisten Diskussionen, welche im Internet geführt werden, sind Foren eine beliebte Plattform. Hier können auf einfachen Weg neue Fragen gestellt, bestehende Fragen erweitert oder beantwortet werden. Da für das Schreiben in Forum zum Großteil Pseudonyme verwendet werden, ist außerdem eine problemlose Beteiligung an Diskussionen von zum Beispiel Studenten möglich die sich in einer Vorlesung nicht trauen Fragen zu stellen. Ebenfalls für Personen die sich nicht trauen bestimmte Fragen zu stellen, weil sie diese für zu dumm halten, sind Foren, Blogs oder Wikis ein guter Anlaufpunkt. In diesen Fall kann eine oftmals vorhandene Suchfunktion benutzt werden, um nach dieser oder ähnlichen Fragen zu suche die andere zuvor gestellt beziehungsweise beantwortet habe.

In der Regel gibt es für ein Thema nicht nur eine sondern einen große Anzahl verschiedener Orte die ein bestimmtes Thema behandeln. Im Fachbereich Informatik der Technischen Universität Darmstadt existiert zum Beispiel von der Fachschaft betriebenes Forum mit Unterforen zu den verschiedensten Veranstaltungen. Gleichzeitig wird nebenbei das LMS Moodle[2] eingesetzt, in dem zu einigen Vorlesungen eigne Foren betrieben werden. Hierbei kann es passiert, dass einzelne Diskussionen nur auf einer der beiden Plattformen geführt werden und möglicherweise wichtige

[1] `http://www.facebook.com`
[2] `https://moodle.org/`

Beiträge verpasst werden. Im umgedrehten Fall werden Diskussionen mehrfach an verschiedenen Orten geführt, dadurch entsteht eine Redundanz der gleichen Informationen. Will man hierbei zur Vermeidung auf weitere Quellen (Beiträge, Vortragsfolienfolien, ...) verweisen, bleibt im Normalfall nur die Möglichkeit dies schriftlich in der Form „schau bei Veranstaltung x in Foliensatz y auf Folie z“ zu verweisen. Eine richtige Verknüpfung findet hier in den seltensten Fällen statt.

2 Grundlagen

2.1 Datenintegration

2.1.1 Semantic web und das Resource Description Framework

Idee semantic web einbauen

Eine der bekanntesten Umsetzungen der Vision des semantischen Webs ist wohl das Resource Description Framework (RDF). Wie der Name schon suggeriert dient RDF zur Beschreibung von einzelnen Ressourcen innerhalb des Internets. Nach [7, 8] bestand die Motivation bei der Entwicklung von RDF Information über Ressource in einen offenen Datenmodell zu speichern, so dass diese Daten von Maschinen automatisch verarbeitet, manipulieren und untereinander ausgetauscht werden können. Gleichzeitig sollte es auch einfach von jedem erweitert werden können „RDF is designed to represent information in a minimally constraining, flexible way“[7].

Das Datenmodell von RDF ist sehr einfach aufgebaut um es effizient verarbeiten zu können. Die Grundlage bilden simple Tripel aus Subjekt, Prädikat und Objekt, welche an die natürliche Sprache angelehnt als Sätze[5] bezeichnet werden. Mehrere solcher Tripel bilden einen RDF Graphen. Das Prädikat beschreibt hierbei eine Beziehung zwischen Subjekt und Objekt und wird daher oft als Eigenschaft beschrieben. In der natürlichen Sprache kann man dies zum Beispiel so ausdrücken; Das Subjekt hat eine Eigenschaft mit den Wert Objekt. Graphisch wird dies durch eine gerichtete Verbindung von Subjekt und Objekt mit der Beschriftung der Prädikats dargestellt (siehe Abbildung 2.1).

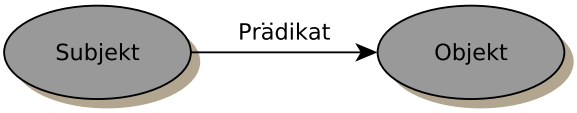

Abbildung 2.1.: Graphische Darstellung eines RDF Tripels

Für Subjekt, Prädikat und Objekt besteht die Möglichkeit sogenannte Uniform Resource Identifiers (URI), Literale oder leere Knoten als Werte einzusetzen.

URIs sind eindeutige Bezeichner die eine beliebige reale oder abstrakte Ressource und werden wie in RFC 2396[1] beschrieben formatiert, wobei relative URIs nach [7] nicht verwendete werden sollen.

Literale bestehen aus einfachen Zeichenketten die zum Speichern der Informationen dienen. Zusätzlich können Literale mit der Angabe der verwendeten Sprache `"Objekt"@de` oder

[1] `http://www.isi.edu/in-notes/rfc2396.txt`

des Datentyps `"42"^^xsd:integer` erweitert werden. Bei Literalen ist aber auch darauf zu achten, dass `"Objekt` und `"Objekt"@de` beschreiben zwar beide den gleichen Wert werden aber von RDF nicht als gleich angesehen. Zwei Literale können nur gleich sein, wenn sie die selbe Sprache beziehungsweise den selben Datentyp besitzen.

Leere Knoten werden als alle Knoten im RDF Graphen beschrieben, welche weder eine URI noch ein Literal sind. Sie dienen häufig dazu um Subjekte zu beschreiben für die aber nicht unbedingt eine eigene URI nötig ist und sind nur innerhalb eines Graphen eindeutig

Beispiel

.

Doch nicht jeder davon ist in jeden Teil des Tripels erlaubt. Das Subjekt ist entweder eine URI oder ein leerer Knoten wobei das Prädikat nur eine URI sein kann. Dahingegen ist es beim Objekt möglich eine URI, einen leeren Knoten oder ein Literal zu verwendeten.

RDF Darstellungsformate

RDF/XML

N3 ist die Kurzform für Notation 3 und wurde von Tim Berners-Lee als Sprache für RDF entwickelt. Tripel in N3 werden dabei wie Sätze in meisten natürlichen Sprachen geschrieben. Erst das Subjekt, dann das Prädikat und am Ende das Objekt gefolgt von einen Punkt, wie in Listing 2.1 zu sehen ist.

Listing 2.1: Einfaches N3 Beispiel

```
1 <http://example.de/florian> <http://example.org/#name> "Florian"
    .
2 <http://example.de/florian> <http://example.org/#age> "28" .
```

Die URI `http://example.de/florian` beschreibt hierbei eine Ressource welche einen Namen `Florian` und ein Alter `28` besitzt. Es ist darauf zu achten, dass alle URI immer zwischen spitzen Klammern stehen. Da nun einzelne Prädikate recht häufig innerhalb eines Graphen auftauchen können, kann es einfach sein diese abgekürzt schreiben zu können. Hierzu ist es möglich sogenannte Präfixe (auch Namensräume genannt) am Beginn des Dokumentes zu definieren und einen so Schreibarbeit abzunehmen.

Listing 2.2: Präfixe

```
1 @prefix person: <http://example.org/#> .
2 <http://example.de/florian> person:name "Florian" .
3 <http://example.de/florian> person:age "28" .
```

Am Anfang von Listing 2.2 wird durch Einleiten mittels des Schlüsselwortes `@prefix` ein neuer Präfix `person:` für die URI `http://example.org/#` festgelegt (man beachte wieder den Punkt am Ende der Zeile). Dieser Präfix kann nun überall innerhalb des Dokumentes verwendet werden, wobei die spitzen Klammern der vorherigen URI weggelassen werden können.

Leere Knoten + kurz Turtle

Abfragesprache SPARQL

Ontologien

2.2 Semantic Interlinked Online Communities

Semantic Interlinked Online Communities[2] (SIOC, ausgesprochen „schock“) ist ein Projekt, welches von Uldis Bojārs und John Breslin begonnen wurde um unterschiedliche, webbasierte Diskussionslattformen(Blog, Forum, Mailinglist,...) untereinander verbinden zu können [2]. Der Kern von SIOC besteht aus einer Ontologie, welche den Inhalt und die Struktur diese Plattformen in ein maschinenlesbares Format bringt und es erlaubt diese auf semantischer Ebene zu verbinden. Auch soll es so möglich sein Daten von einer Plattform zu einer Anderen zu transferieren und so einfacher Inhalte austauschen zu können. Als Basis für SIOC dient RDF, die Ontologie selber wurde in RDFS und OWL designt. Um nicht das Rad neu erfinden zu müssen greift SIOC auf schon bestehende und bewährte Ontologien zurück. Für die Abbildung von Beziehungen zwischen einzelnen Personen wird Friend of a Friend[3] (FOAF) und für einige Inhaltliche- und Metadaten (Titel, Inhalt, Erstelldatum, ...) Dublin Core Terms[4] eingesetzt.

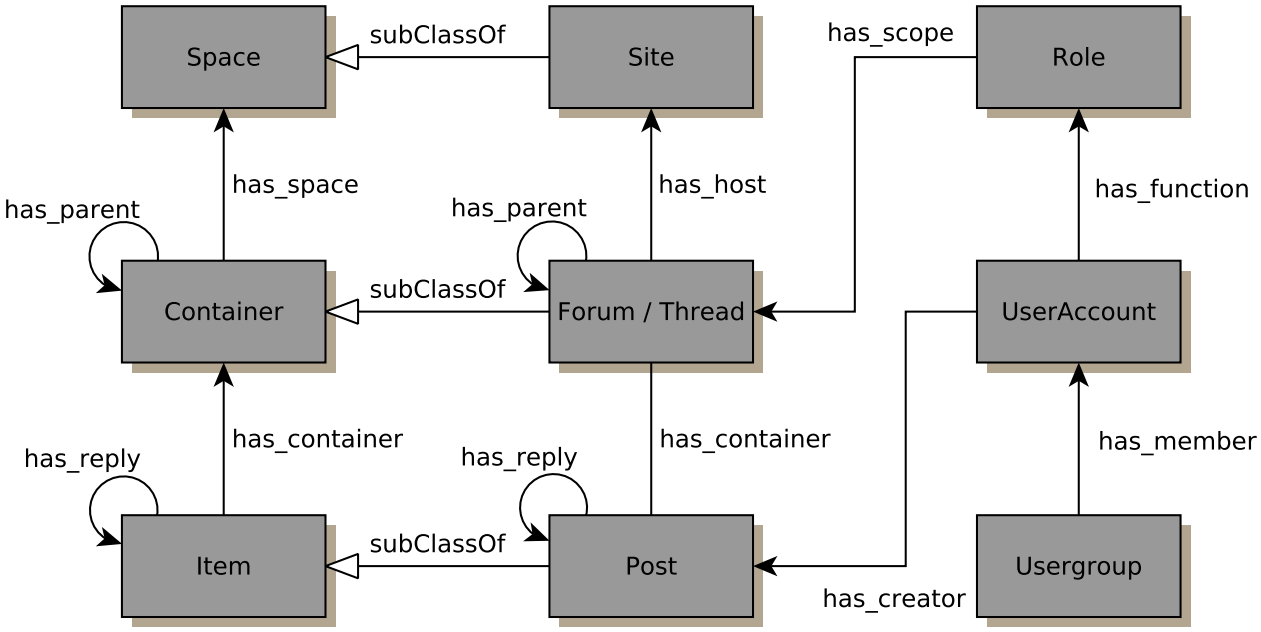

Abbildung 2.2.: Aufbau von SIOC (modifiziert) - Originalquelle: [3]

2.3 Datenverteilung

[2] http://sioc-project.org/
[3] http://www.foaf-project.org/
[4] http://dublincore.org/documents/dcmi-terms/

2.3.1 Enterprise Integration Pattern

2.3.2 Java Messaging Service

2.3.3 Apache Camel

2.4 Verwandte Arbeiten und Projekte

2.4.1 Reclaim Social

Hat sich nicht jeder schon einmal vor den Rechner gesessen um, zum Beispiel, nach einen Bild gesucht das man irgendwann auf irgendeinem der unzähligen sozialen Netzwerke hochgeladen hat, einem aber partout nicht einfallen will wo? Wann und wo habe habe ich den Beitrag geschrieben, der perfekt zu meiner aktuellen Arbeit passen würde? Solche oder ähnliche Fragen wurden sicherlich schon mehrere Millionen mal von verschiedenen Menschen in der Welt des Internets gestellt. Wer hätte in so einen Fall nicht gerne alles was man über die letzten Jahre an verschiedenen Stellen im Netz geschrieben, hochgeladen oder als für ihn wichtig markiert hat zentral gespeichert um es durchsuchen zu können? Genau diesem Thema haben sich Sascha Lobo und Felix Schwenzel angenommen und auf der Netzkonferenz re:publica[5] 2013 ihr gestartetes Projekt „Reclaim Social“ [9] vorgestellt.

Ziel mit diesem Projektes soziale Medien aus allen möglichen Quellen auf seinen eigenen Blog zu spiegeln und so einen zentrale Anlaufstelle für seine eigenen Inhalte schaffen. Aufbauend auf der weit verbreiteten Blogsoftware „WordPress[6]“ und der dafür vorhandenen Erweiterung „FeedWordPress“[7]. Diese Kombination ermöglichst alle Internetseiten, welche einen RSS Feed[8] anbieten, in die Datenbank von WordPress zu spiegeln. Das Problem hierbei besteht darin, dass einige sehr beliebte Internetseiten solche RSS Feeds nicht anbieten (`https://facebook.com`, `https://plus.google.com`) oder eingestellt haben (`https://twitter.com`). Für einige solcher Seiten wurden „proxy-scripte“[9, Tecg Specs Details] implementiert, welche für diese einen RSS Feed emulieren. Zugleich können in den Feeds enthaltende Medien, wie Bilder und Videos(bisher nur als Referenz), heruntergeladen und in WordPress gespeichert werden. So ist es möglich alle gespiegelten Daten einfach zu durchsuchen oder nach bestimmten Kriterien zu filtern. Zusätzlich können alle Freunde, welche auch Reclaim Social einsetzen, in einen Kontaktliste eingetragen und so auch deren Inhalte eingebunden werden.

Aktuell befindet sich dieses Projekt noch im Alpha Stadium und die Installation ist relativ kompliziert. Es ist aber geplant eine eigene Erweiterung für WordPress zu schreiben „he goal is to build just one Reclaim Social-plugin for any wordpress user“[9, How Does It Work]

[5] `http://re-publica.de/`
[6] `http://wordpress.org/`
[7] `http://feedwordpress.radgeek.com/`
[8] `http://www.rssboard.org/rss-specification`

3 Analyse

Inhalt: wass gibt es, was muss neu gemacht werden

Begriff soziales Netzwerk anpassen

Um sich eine Vorstellung davon zu machen, wie ein System auszusehen hat und welche Komponenten dazu nötig sind um zwei oder mehrere sozialen Netzwerke zu verbinden, soll hierzu ein kleines Ablaufbeispiel konstruiert werden.Das vollständige Ablaufdiagramm befindet sich im Anhang A.1.

3.1 Neuen Beitrag verfassen

Alles beginnt damit, dass zum Beispiel ein Student im sozialen Netzwerk A, im Forum zur Veranstaltung Telekooperation 1, eine Frage zur aktuellen Übung stellen will. Er geht zuerst in den passenden Thread und beginnt einen neuen Beitrag zu schreiben. Sobald er fertig ist, klickt er auf „Absenden“ und sein Beitrag wird in der Datenbank des sozialen Netzwerkes A gespeichert und als neuer Eintrag im Thread angezeigt (siehe Abbildung 3.1).

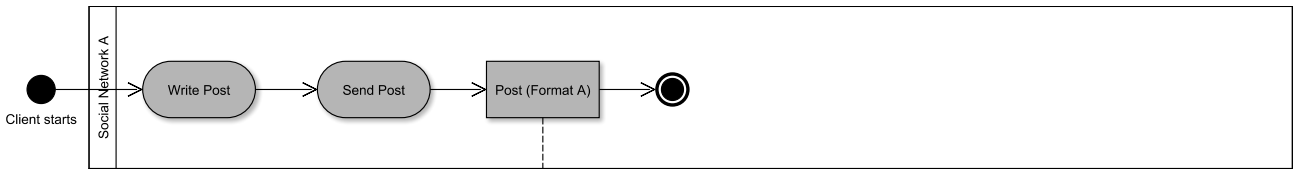

Abbildung 3.1.: Benutzer erstellt einen Beitrag im sozialen Netzwerk A.

3.2 Beiträge von sozialen Netzwerk A lesen

Um diese Beitrag in das soziale Netzwerk B transferieren zu können, müssen zuerst die Daten über eine öffentliche Schnittstelle vom Server des sozialen Netzwerks A heruntergeladen werden. Da in der Regel nicht automatisch bekannt ist, wann ein neuer Beitrag vorhanden ist, müssen die Server in zeitlichen Abständen abgefragt (polling genannt) und die zurückgelieferten Daten nach neuen Beiträgen durchsucht werden. Sind ein oder mehrere neue Beiträge gefunden worden, können diese nicht direkt an das soziale Netzwerk B geschickt werden, da sich diese in der Regel im verwendeten Datenformat unterscheiden. Diese müssen zuvor konvertiert werden.

Die einfachste Möglichkeit wäre nun die Daten von Format A nach Format B zu konvertieren. Bei zwei Formaten ist dies noch sehr einfach. Es müsste lediglich ein Konverter von Format A nach Format B und einer in die umgekehrte Richtung implementiert werden. Für den Fall, dass nun ein weiteres Netzwerk C unterstützt werden soll, würde ich die Anzahl an nötigen Konvertern auf Sechs erhöhen, wie Tabelle 3.1 zeigt.

Tabelle 3.1.: Anzahl Konverter bei drei sozialen Netzwerken

		Nach		
		Netzwerk A	**Netzwerk B**	**Netzwerk C**
Von	**Netzwerk A**	-	×	×
	Netzwerk B	×	-	×
	Netzwerk C	×	×	-

Nimmt man an n_{sn} sei eine beliebige Anzahl sozialer Netzwerke, entspricht die Anzahl der notwendiger Konverter $n_{k1} = n_{sn} * (n_{sn} - 1)$, da für jedes Netzwerk ein Konverter in alle anderen Netzwerke erzeugt werden muss. Sollen nur Zwei oder Drei Netzwerke unterstützt werden ist der Aufwand noch sehr überschaubar, bei mehr kann dies aber sehr Aufwendig werden.

Eine elegantere Methode, welche die Anzahl zu implementierender Konverter in Grenzen halten kann, wäre die Einführung eines Zwischenformates. Geht man davon aus, dass die Daten aller Netzwerke nur in dieses Zwischenformat geschrieben und aus diesem gelesen werden müssen, würde sich der Aufwand auf maximal Zwei Konverter pro neuem Netzwerk reduzieren. Für eine beliebige Anzahl Netzwerke wären also $n_{k2} = n_{sn} * 2$ Konverter nötig. Nachteile hätte dieser Ansatz nur für $n_{sn} = 2$ und $n_{sn} = 3$, da in diesen Fällen mehr beziehungsweise gleich viele Konverter gegenüber der ersten Methode erforderlich wären. Erhöht man die Anzahl Netzwerke jedoch nur geringfügig, sinkt die Menge an Konvertern sichtbar. Für $n_{sn} = 4$ wären es $n_{k2} = 8$ statt $n_{k1} = 12$ und für $n_{sn} = 5$ ergibt sich $n_{k2} = 10$ statt $n_{k1} = 20$ Konvertern. Gleichzeitig können so syntaktische Unterschiede in den einzelnen Formaten angeglichen werden, was sie leichter handhabbar macht. Abbildung 3.2 verdeutlicht den Ablauf unter Verwendung des eben beschriebenen Zwischenformats.

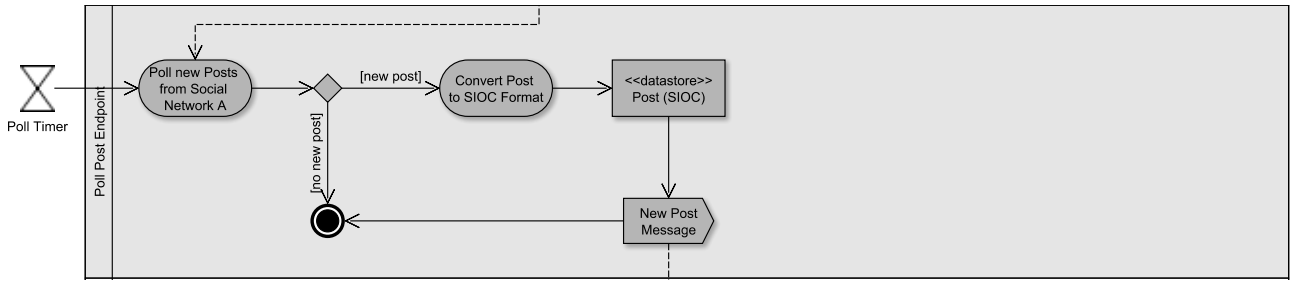

Abbildung 3.2.: Lesen des erstellten Beitrags und konvertieren in das Zwischenformat.

Liegen nun alle Daten in diesem Zwischenformat vor, könnten diese an einer Datenbank gespeichert werden. Diese Datenbank macht es möglich auf einfache Art und Weise auf die gespeicherten Daten von außerhalb zu zugreifen und diverse Abfragen ausführen zu können. Dies könnte zum Beispiel sein, in den Daten nach inhaltlich gleichen oder ähnlichen Beiträgen zu suchen oder passende externe Materialien aus Vorlesungen zu einen Thema vorzuschlagen. Die Möglichkeiten sind hier vielfältig.

Da ein Teil dieser Arbeit darin besteht Beiträge zwischen zwei oder mehr sozialen Netzwerken zu synchronisieren, muss das Eintreffen neuer Beiträge an die einzelnen Komponenten mitgeteilt werden. Da es unmöglich ist diesen Zeitpunkt vorher zu sagen ist der beste Weg einen ereignisorientierten Ansatz zu wählen. Hierbei wird beim lesen eines neuen Beitrags eine Ereignisnachricht erzeugt, welche die interessierten Komponenten über das Eintreffen informiert. So wird eine

zeitliche Entkopplung von Lesen und Schreiben möglich, gleichzeitig können sich einzelnen Komponenten an diesen Nachrichtenstrom an- oder abhängen ohne Andere zu stören. Dies erhöht die Flexibilität des Systems.

3.3 Beitrag in soziales Netzwerk B schreiben

Hat sich eine Komponenten und empfängt eine Ereignisnachricht von einen neuen Beitrag, wird ähnlich Verfahren wie schon beim Lesen, nur umgekehrt (Siehe Abbildung 3.3 links, oberer Ablauf). Der neue Beitrag wird vor den schreiben in das soziale Netzwerk B aus dem Zwischenformat in der Zielformat B konvertiert. Alle dazu nötigen Information können direkt aus der Ereignisnachricht oder zusätzlich aus der oben genannten Datenbank geholt werden.

Wünschenswert wäre es hierbei, wenn es so aussehen würde der Beitrag nicht vom hier entwickelten System, sondern von Autor des ursprünglichen Beitrags geschrieben worden. Hierzu es unerlässlich auf das Konto des Autor im entsprechenden Netzwerk zugreifen zu können, da im allgemeinen der Schreiben stellvertretend für dritte Personen nicht möglich ist. Hierzu muss zunächst für den betreffenden Autor das Konto für das soziale Netzwerk B herausgesucht werden (Siehe Abbildung 3.3 links, unterer Ablauf). Wurde ein passendes Konto gefunden, wird der konvertierte Beitrag über diesen geschrieben. Sollten kein Passender gefunden werden, müsste dies über ein vorher definiertes Konto geschehen. Dabei sollte aber auf den ursprünglichen Autor beziehungsweise Beitrag in irgendeiner Form hingewiesen werden. Dies kann zum Beispiel durch anbringen eines Links an den Text des Beitrags erfolgen.

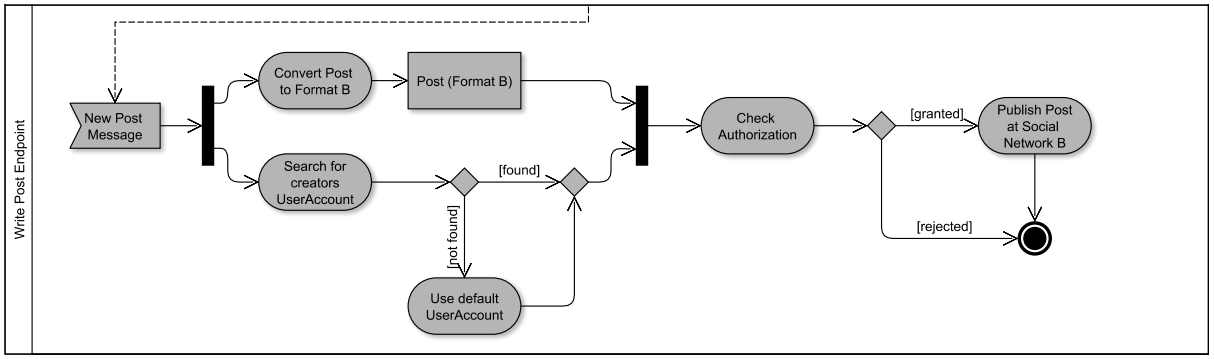

Abbildung 3.3.: Konvertierten des Beitrags in das Format B und schreiben in das soziale Netzwerk B

Gleichzeitig mit den Sammeln von Daten kommt immer auch das Thema zum Schutz der Privatsphäre auf. Wie soll damit umgegangen werden, wen ein Benutzer nicht möchte dass seine Beiträge gesammelt werden oder automatisch weiter geleiten werden sollen? Hierzu wäre es sinnvoll die eben beschriebenen Abläufe so zu erweitern, dass der Benutzer festlegen kann bestimme Quellen oder einzelne Teile für das automatische Lesen und Schreiben zu blockieren, wie es von Uldis Bojars et al. in [1] vorgeschlagen wird. Dies könnte durch eine Access Control List (ACL) realisiert werden. Hier könnte der Benutzer Lese und Schreibrechte für einzelne Bereiche festlegen, welche dann das System benutzt um einzelne Abläufe auszuführen oder abzubrechen.

3.4 Identifizierung der Komponenten

Anhand dieses kurzen Ablaufbeispiels können wir nun einige Komponenten ablesen die das System unbedingt beinhalten muss und welche ergänzend dazu Wünschenswert wären:

- Eine Komponente muss Daten von einen sozialen Netzwerk in das System einlesen und diese in ein geeignetes Zwischenformat konvertieren können.
- Eine weitere Komponente nimmt Beiträte im Zwischenformat entgegen, konvertiert diese in das Format des entsprechenden Netzwerkes und schreibt diese dorthin.
- Um stellvertretend für einen Benutzer schreiben zu können muss es möglich sein nach Konto eines Benutzer in einem Netzwerk suchen zu können.
- Um die Privatsphäre der Benutzer zu wahren, wäre eine ACL Mechanismus sinnvoll.

4 Eigener Ansatz

- Probleme und deren Lösung während der Umsetzung
- Beispiel: Zugriff auf Moodle über Webservice

4.1 Social Online Community Connectors

4.1.1 Datenformat

4.1.2 Konfiguration

Eine wichtige Designentscheidung der Connectoren war es, wo alle zum Betrieb notwendigen Einstellungen und zusätzliche Daten gespeichert werden sollen? In den ersten Testimplementierungen wurden diese Daten teilweise in einfachen Dateien, nur für die anfallenden Daten im RDF Format war von Beginn an eine Speicherung in einem RDF Triplestore vorgesehen. Im späteren Verlauf dieser Arbeit kam die Idee auf komplett alle Daten dort zu speichern. Da es vorkommen kann, dass bestimmte Einstellungen wie Anmeldedaten mehrfach genutzt werden und im Falle eine Änderung müssten die Daten nur an diesem einen Ort geändert werten. So könnten auch diverse Einstellungen über verschiedene unabhängige System hinweg genutzt werden.

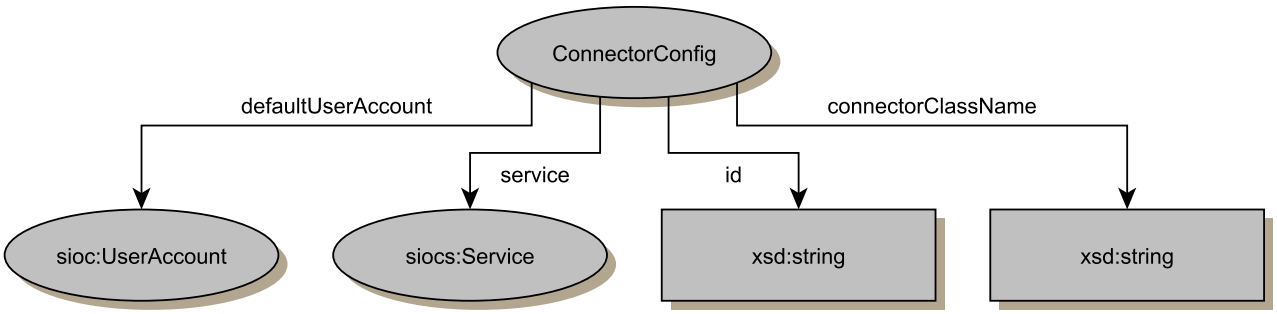

Abbildung 4.1.: Connector Config Ontology

Aus diesem Grunde wurde für die Konfiguration eines Connectors die *Connector Config Ontology* entwickelt. Abbildung 4.1 zeigt diese Ontologie. Jeder Connector bekommt einen eindeutigen Bezeichner(kurz ID) um jeden diesen identifizieren zu können. Dies kann eine beliebige Zeichenkette sein. Um die korrekte Javaklasse des zu erstellenden Connectors benutzen zu können, wird der vollständige Klassenname, wie er zum Beispiel von `object.class.getName()` zurückgegeben wird, in `connectorClassName` gespeichert. Für den Zugriff auf die Daten der SOC sind bei einiges APIs bestimmte Parameter wie die genaue Adresse des Servers notwendig. SIOC bietet hierzu ein eigenes Modul an, welches genau für solche Beschreibungen von Diensten die SIOC

Ontologie erweitert. Eine kurze Beschreibung dieses Moduls und eine Erweiterung des selbigen für den Einsatz in SOCC befindet sich im Abschnitt 4.1.2 und **??**. Als Letztes braucht ein Connector noch einen vordefinierten Benutzer welcher als `defaultUserAccount` als SIOC UserAccount gespeichert wird. Dieser vordefinierte Benutzer erfüllt im Großen und Ganzen zwei Aufgaben. Als Erstes wird er für lesende Zugriffe der API auf die SOC genutzt, da hierzu ein einzelner Benutzer vollkommen ausreichend ist. Die zweite Aufgabe bezieht sich auf das stellvertretende Schreiben einzelner Benutzer. Nicht immer werden die dazu notwendigen Daten von den Benutzer zur Verfügung gestellt oder sind unbekannt. In diesem Fall wird der vordefinierte Benutzer genutzt und der Beitrag mit einem Vermerk zum original Autor über diesen geschrieben.

Services

Wie eben schon beschrieben, existiert für SIOC ein Modul zur einfachen Modellierung von Diensten auf semantischer Ebene. Kernstück dieses Moduls ist die Klasse Service, wie auf Abbildung 4.2 zu sehen ist. Mit dieser Klasse kann durch eine Hand voll Eigenschaften ein Dienst beschrieben werden. Für diese Arbeit ist davon die wichtigste Eigenschaft `service_endpoint`. Durch diese kann die Adresse festgelegt werden, unter dem ein bestimmter Dienst erreichbar ist. Gerade bei Plattformen die nicht an eine feste Adresse (Foren, Blogs, ...) gebunden sind, ist diese Angabe unerlässlich.

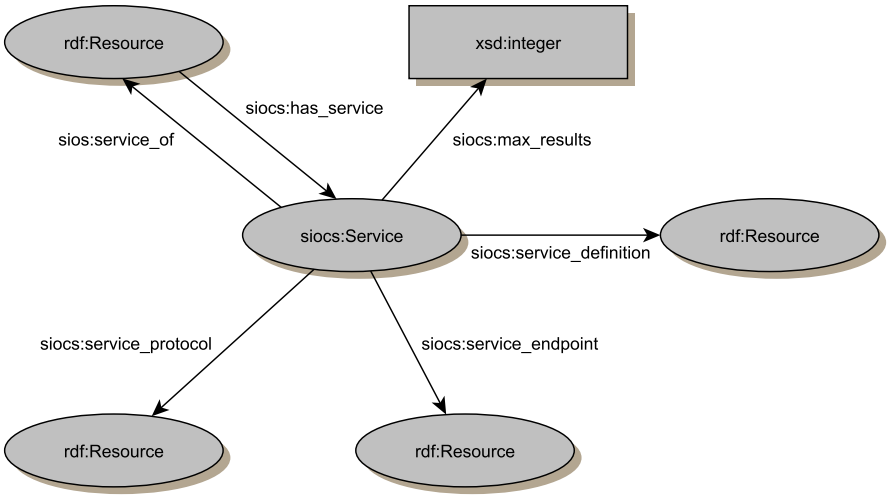

Abbildung 4.2.: SIOC Services Ontology

Die Eigenschaften `has_service` und `service_of` sind sind ideal zur Verbindung von einzelnen SIOC UserAccounts mit einem Service. Diese Verbindung hilft dabei für das stellvertretende Schreiben von Beiträgen schnell die passenden Benutzerdaten zu finden. Ebenfalls nützlich ist `max_results`. Manche Dienste erlauben es nur eine maximale Anzahl an Ergebnissen pro Aufruf zurückgeben zu lassen. Da sich diese Anzahl über die Zeit ändern kann ist es nicht sinnvoll diese fest im Programm festzulegen, kann diese so im Nachhinein verändert werden. Für SOCC weniger interessant aber Vollständigkeit halber seien noch erwähnt `service_protocol` zum Angeben des

verwendeten Übertragungsprotokolls (REST[1], SOAP[2], ...) und `service_definition` mit dem auf eine weiterführende Definition verwiesen werden kann.

UserAccounts

Autorisierung und Authentifizierung

Die Wahl für SIOC als Datenformat zu hat sich nach den ersten Tests als eine sehr gute Entscheidung herausgestellt. Mittels SIOC ließen die wichtigsten Daten aller untersuchten Plattformen in einer einheitlichen Form speichern. Die ersten Probleme traten erst auf, als es daran ging Daten für die Autorisierung (Feststellen ob jemand eine Handlung ausführen darf) beziehungsweise Authentifizierung (Feststellen ob jemand der ist, den er vorgibt zu sein) an den verschiedenen Programmierschnittstellen einen geeigneten Ort zu finden. Dazu musste aber erst festgestellt werden, welche verschiedenen Daten alle vorliegen können.

Username/Passwort ist wohl eine der ersten und häufigsten Mechanismen, um den Zugriff sensibler Daten vor Dritten zu schützen. Das in Abschnitt 5.1.1 beschriebene LMS Moodle, setzt zum Beispiel den Username und Password eines angemeldeten Benutzers zu Authentifizierung ein.

OAuth [3] stellt heutzutage den Standard der verwendeten Authentifizierungsmechanismen für hauptsächlich webbasierte API dar. Benutzer können so temporär Programmen den Zugriff auf ihre Daten erlauben und später wieder verbieten. Der aktuelle Standard stellt OAuth 2.0 dar und wird in dieser Version von den größten Seitenbetreibern wie Google, Facebook oder Microsoft eingesetzt[4]. Insgesamt sind für die Nutzung von OAuth vier Parameter wichtig. Für das Programm, dass Zugriff erhalten möchte sind die Parameter *client_id* und *client_secret* [4][S. 8]. Sie weisen das Programm als autorisiert für die Benutzung der Schnittstelle aus. Soll nun beim Aufrufer einer von OAuth geschützten Funktion belegt werden ist ein sogenannter Accesstoken[4][S. 9] nötig. Da dieser Accesstoken in der Regel nur eine bestimmte Zeit gültig ist, wird je nach Implementierung des Standards noch ein Refreshtoken mitgeliefert. Mit diesem Refreshtoken ist das Programm in der Lage ohne Zutun des Benutzers einen abgelaufen Accesstoken wieder zu aktivieren. Dies kann beliebig oft wiederholt werden, bis der Benutzer beide Token für ungültig erklärt.

vll. noch OAuth 1.0(a) einbauen

API Schlüssel sind eine dritte Möglichkeit Programmen Zugriff auf eine API zu gewähren. Der API Schlüssel entspricht ungefähr einer Kombination von client_id und client_secret von OAuth. Dieser Schlüssel schaltet in der Regel nicht den Zugriff auf persönliche Daten von Benutzer frei. Hier ist noch ein weiterer Mechanismus wie die Verwendung von einem Usernamen und Passwort nötig. Die in Abschnitt 5.1.4 beschriebene Google Youtube API hierzu ein gutes Beispiel.

[1] Representational State Transfer
[2] Simple Object Access Protocol
[3] `http://oauth.net/`
[4] `http://en.wikipedia.org/wiki/OAuth#List_of_OAuth_service_providers`

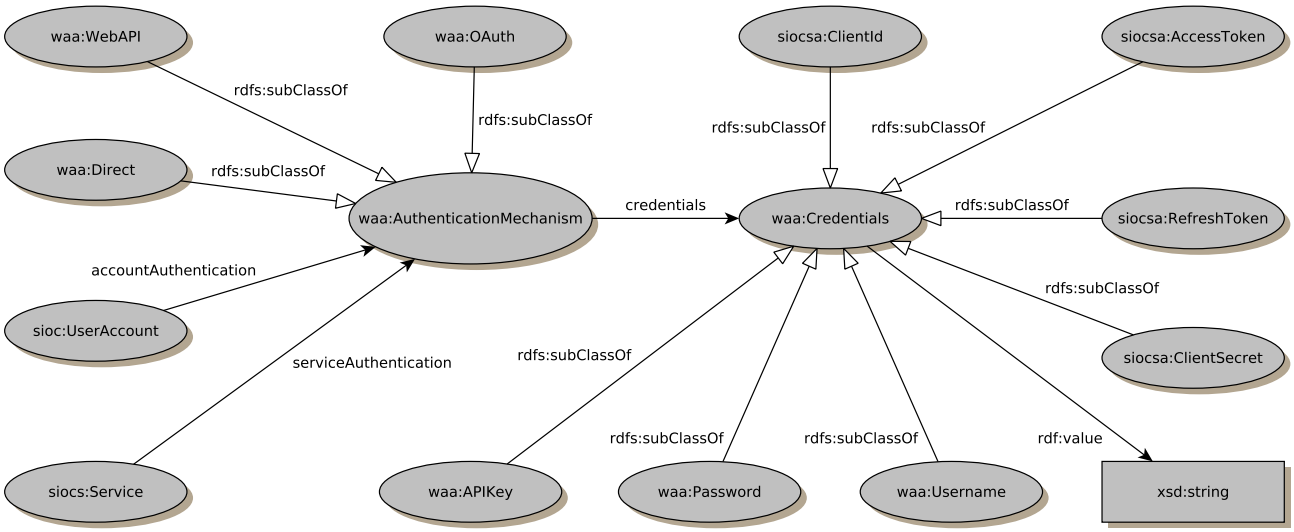

Abbildung 4.3.: SIOC Services Authentication Ontology

Neben diesen drei Mechanismen wäre noch der Vollständigkeit halber die HTTP-Authentifizierung zu nennen. Hierbei handelt es ich um eine Form des Username/Passwort Verfahrens, welches auf das HTTP Protokoll aufsetzt. Für einfachen Webseiten ist dies ein unkomplizierte Art die Datei vor fremden Zugriffen zu schützten. Für aktuelle öffentliche APIs ist diese Form der Authentifizierung nicht mehr Stand der Technik.

Die Suche nach einer bestehen Ontologie, welche zusammen mit SIOC verwendet werden könnte, gestaltete sich als sehr schwierig. Ein Großteil der Ontologien in diese Richtung befasst sich eher mit dem Thema der Autorisierung wie Zugriffssteuerungsliste wie zum Beispiel die *Web Access Control List* [6]. Nicht desto trotz existiert auch eine Authentifizierung Ontologie. Die *Authentication Ontology*[5] des *OmniVoke*[6] ist für die notwendigen Zwecke das ideale Werkzeug.

fertig schreiben

4.1.3 Aufbau eines Connectors

ClientManager

StructureReader

Um auf Informationen über die Struktur von Foren, sozialen Netzwerken und so weiter im SIOC Format zugreifen zu können, implementiert jeder Connector dazu einen StructureReader. Die Struktur lässt sich, wie im Abschnitt 4.1 gezeigt, durch die SIOC Klassen *Site* und *Container* (und Unterklassen davon) beschrieben.

Abbildung 4.5.: StructureReader

[5] http://omnivoke.kmi.open.ac.uk/authentication/
[6] http://omnivoke.kmi.open.ac.uk/framework/

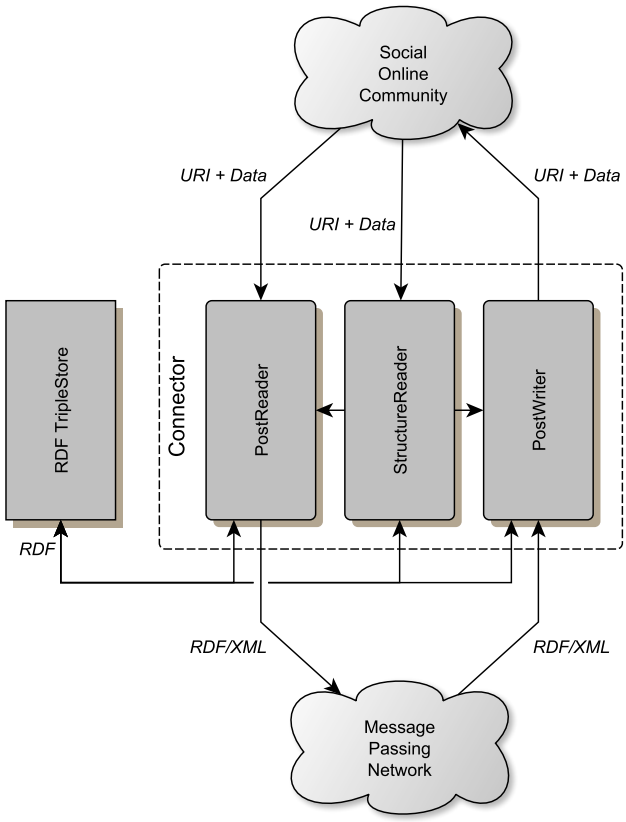

Abbildung 4.4.: Übersicht der Komponenten der SOCC

Um auf diese Struktur zugreifen zu können, enthält die StructureReader Schnittstelle mehrere Methode (Siehe Abbildung 4.5).

`getSite()` ist eine Methode, welche die Beschreibung einer Seite (Forum, Blog, soziales Netzwerk) als SIOC Site Objekt zurücklieft. Dies wird relativ häufig um die Zugehörigkeit einiger Objekte durch einen Link zu dieser Seite zu verdeutlichen. Dies kann bei einigen APIs nützlich sein, da dort manchmal keine Information zum *Container* eines Beitrags mitgeliefert werden, über den man sonst eine Beziehung zwischen Seite und Beitrag herstellen könnte.

`getContainer(URI)` ist dazu gedacht die Information eines einzelnen Containers erhalten der sich hinter eine URI verbirgt.

`listContainer(...)` sind Methoden welche für den die Auflisten aller Container einer Seite zur Verfügung stehen. Die Methode ohne Parameter listet alle Container auf der ersten Ebene auf. Dies könnten zum Beispiel alle Kurse auf einen Canvas LMS Seite oder alle Gruppen auf Facebook sein. Die zweite Methode mit URI Parameter gibt eine Liste alle Container, welche den Container hinter der übergeben URI als Elternteil haben, zurück. Als Beispiel wären alle Themen innerhalb eines Forums zu nennen.

`hasChildContainer(URI)` überprüft ob der Container hinter einer URI überhaupt weitere Container als Kinder besitzt. Diese Methode wird dazu eingesetzt, um vorab zu testen, ob der Aufruf von `listContainer(URI)` das gewünschte Ergebnis liefert oder ein Fehler auftritt.

PostReader

Der `PostReader` dient als Schnittstelle, um auf geschriebene Beitrage innerhalb eines Containers oder die Kommentare auf eines Beitrags zuzugreifen. Die Methode `hasPosts(URI)` ist nur zur Überprüfung ob ein Container oder ein Beitrag hinter einer URI weiter Beiträge enthält die gelesen werden können. Um einen einzelnen Beitrag anhand seiner URI lesen zu können kann die Methode `getPost(URI)` genutzt werden. Sei liefert dann den Beitrag SIOC Post Objekt zurück beziehungsweise einen Fehler, falls der Beitrag nicht mit diesem Connector gelesen werden kann. Die wichtigste Methode dahingegen ist `pollPosts(URI, Date, int)`. Insgesamt können dieser Methode drei Parameter übergeben werden. Der Erste ist eine URI die den Ort angibt von der alle Beiträge gelesen werden können. Mit dem zweiten Parameter kann ein Zeitpunkt angegeben werden, ab dem ein zu lesender Beitrag geschrieben sein muss. Alle Beiträge die vor diesem Zeitpunkt erstellt wurden, werden nicht zurück gegeben. Der letzte Parameter gibt eine obere Schranke an wie viele Beiträge maximal pro Aufruf dieser Methode gelesen werden dürfen.

Abbildung 4.6.: PostReader

PostWriter

In Abbildung 4.7 ist ein Sequenzdiagramm der PostWriter Komponente zu sehen. Dort ist visualisiert, welche Schritte für das stellvertretende Schreiben von Beiträgen eines Benutzers unternommen werden müssen. Soll nun ein Beitrag in einer SOC geschrieben werden, wird die Methode `writePost(URI, String, Syntax)` mit dem Zielort als URI, dem Beitrag als serialisiertes RDF Objekt und dem verwendeten Serialisierungsformates aufgerufen. Begonnen wird damit, dass als erstes nach einem UserAccount für den Service des aktuellen Connectors des Beitragautors gesucht. Im Idealfall befindet sich für den UserAccount des Beitragsautors ein Link zu seiner FOAF Person und von ein weiterer Link zum UserAccount für den aktuellen Service. Mit diesem UserAccount kann dann vom ClientManager ein Clientobjekt für die verwendete API angefordert werden. Sollte die Suche negativ verlaufen, steht der vordefinierte Client zur Verfügung. Mit diesem Client, ob mit der des Autors oder dem Vordefinierten, wird im letzten Schritt der Beitrag im von der API verwendeten Format in die SOC geschrieben.

4.2 SOCC-Camel

4.2.1 SoccComponent

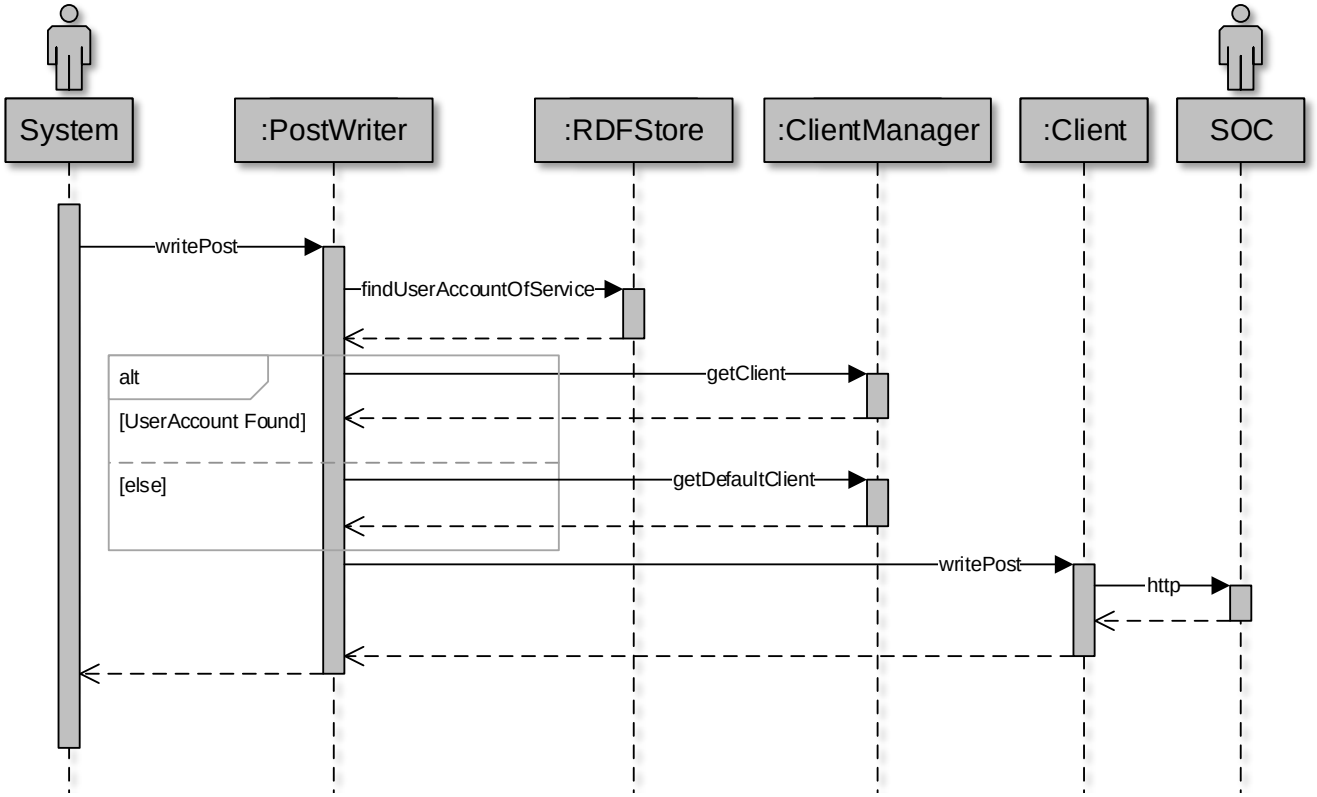

Abbildung 4.7.: PostWriter Sequenzdiagramm

4.2.2 SoccPostPollingConsumer

4.2.3 SoccPostProducer

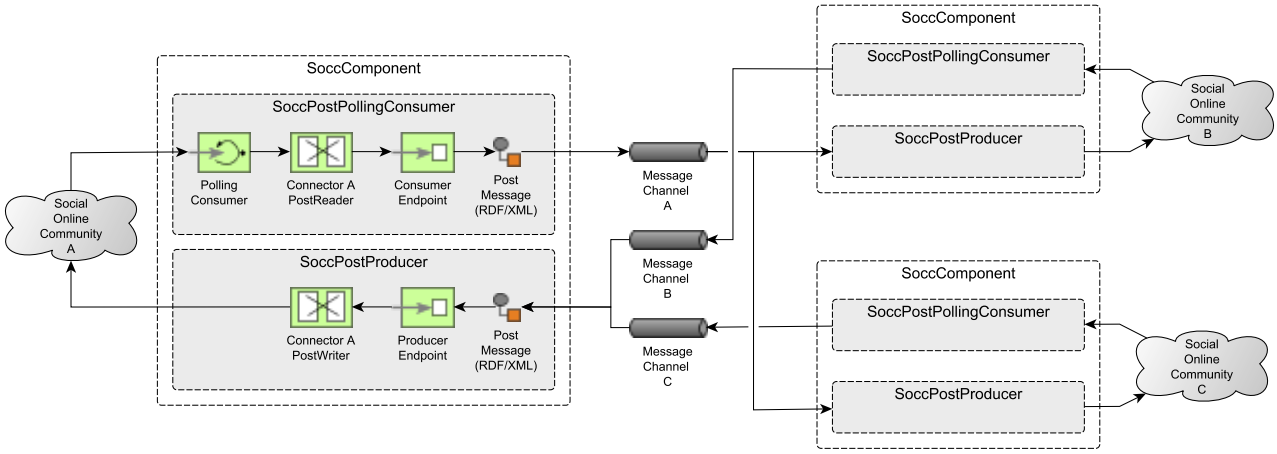

Abbildung 4.8.: Übersicht des Apache Camel Moduls Socc-Camel

5 Implementierung

5.1 Implementierung einiger Connectoren

Das muss rein:

- Mapping nach SIOC
- Zugriff über die API
- Probleme bei der Implementierung

5.1.1 Moodle Connector

- Eingebaute REST Schnittstelle, aber kein Lesen von Beiträgen
- WebService Plugin MoodleWS (REST oder SOAP)
  - https://github.com/patrickpollet/moodlews
  - ClientAPI existieren von selber Autor
  - REST defekt, kein schreiben von Beiträgen möglich
  - SOAP funktioniert mehr oder weniger
  - Verschluckt Fehlermeldungen
  - kein lesen einzelner Posts/Threads/Foren
  - SOAP ClientAPI neu generieren, weil vorhandene nicht mit 2.4 funktioniert.
  - Username/Password + Session Token/Id
  - “Use an auto generated wsdl” -> No
  - schreiben von neuen Beitrag direkt in thread nur als Antwort auf ersten Beitrag möglich
  - Rückgabe aller Beiträge in einem Objekt

5.1.2 Facebook Connector

- REST API + JSON
- keine offizielle Java API für Desktop -> Web + Mobile only
- GraphAPI, Facebook Query Language
- OAuth 2.x
  - kein Refreshtoke

  - Token Haltbarkeit 2h (2 Monate, wen extended)
  - token nur über webbrowser
- RestFB alternative Java API für die REST Schnittstelle der GraphAPI
- Typ der zurückgelieferten Daten nicht anhand der URI erkennbar, häufig erst durch Angabe von *metadata=1*
- beim herunterladen einzelner Posts nicht immer erkennbar wo sie geschrieben wurden

5.1.3 Google+ Connector

- Einfach REST API + JSON
- OAuth
  - Refreshtoken (token laufen quasi nie ab)
  - holen von token ohne webbrowser möglich
- Objekte aufgebaut aus Actor (wer machte was), Verb(wie machte er es), Object (wtas machte er) + Metadata
- verschieden Sprachen + Plattformen
- lesen nur von öffentlichen Beiträgen
- kein Schreiben von Beiträgen

5.1.4 Youtube Connector

- Aktueller Umbau der API (ähnlich google+) v3
  - keine lesen von kommentaren
  - kein schreiben
- alte GData Feed API v2 basiert auf RSS + Youtube Erweiterung
- Mapping teilweise durch basis auf RSS einfach, manchmal auch nicht
- Wichtigen Metadaten nur implizit vorhanden (comment id in uri aber nicht in datenformat)

5.1.5 Canvas LMS Connector

- relativ neues LMS
- super Bedienung
- super REST API
- keine Java API
- rudimentäre Eigenentwicklung einer Java API, Funktionsweise ähnlich G+
- viel API Funktionen wohl nicht extern nutzbar (UserProfil lesen, vll. Falsche Berechtigung -> test nötig)

6 Evaluation

7 Abschlussbetrachtung

7.1 Fazit

7.2 Ausblick

Literaturverzeichnis

[1] Uldis Bojars, John G. Breslin, and Stefan Decker. Porting social media contributions with SIOC. *Recent Trends and Developments in Social Software*, 6045:116–122, 2011.

[2] John G. Breslin, Andreas Harth, Uldis Bojars, and Stefan Decker. Towards semantically-interlinked online communities. *The Semantic Web: Research and Applications*, pages 71–83, 2005.

[3] Digital Enterprise Research Institute. SIOC - Semantically-Interlinked Online Communities. `http://sioc-project.org/`, Zugriff: 2013-08-15.

[4] D. Hardt. The OAuth 2.0 Authorization Framework. `http://tools.ietf.org/html/rfc6749`, accessed: 2013-08-30, 2012.

[5] Hans-Werner Heinzen. Primer: Getting into RDF & Semantic Web using N3 Deutsche Übersetzung. `http://www.bitloeffel.de/DOC/2003/N3-Primer-20030415-de.html`, Zugriff: 2013-08-19.

[6] James Hollenbach, Joe Presbrey, and Tim Berners-Lee. Using RDF Metadata To Enable Access Control on the Social Semantic Web. *Proceedings of the Workshop on Collaborative Construction, Management and Linking of Structured Knowledge (CK2009)*, 514, 2009.

[7] Graham Klyne and Jermey J. Carrol. Resource Description Framework (RDF): Concepts and Abstract Syntax. `http://www.w3.org/TR/rdf-concepts/`, Zugriff: 2013-08-19, 2004.

[8] Frank Manola and Eric Miller. RDF Primer. `http://www.w3.org/TR/rdf-primer/`, Zugriff: 2013-08-19, 2004.

[9] Felix Schwenzel and Sascha Lobo. Reclaim Social. `http://reclaim.fm/`, Zugriff: 2013-08-14.

[10] Qiyun Wang, Huay Lit Woo, Choon Lang Quek, Yuqin Yang, and Mei Liu. Using the Facebook group as a learning management system: An exploratory study. *British Journal of Educational Technology*, 43(3):428–438, May 2012.

A Anhang

A.1 Anforderungsanalyse Ablaufdiagramm