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 der Zusammenfassung

Abstract

Content of the abstract

Inhaltsverzeichnis

1. Einleitung **1**

2. Grundlagen **3**
2.1. Zugriff auf Webanwendungen . . . 3
2.1.1. Representational State Transfer (REST) . . . 3
2.1.2. Simple Object Access Protocol (SOAP) . . . 4
2.2. Datenintegration . . . 5
2.2.1. Semantic Web . . . 5
2.2.2. Ontologien . . . 9
2.3. Datenverteilung . . . 11
2.3.1. Java Messaging Service . . . 12
2.3.2. Enterprise Integration Pattern (EIP) . . . 13
2.4. Lernplattformen und soziale Online-Netzwerke . . . 15
2.4.1. Moodle . . . 15
2.4.2. Canvas . . . 16
2.4.3. Youtube . . . 17
2.4.4. Facebook . . . 17
2.4.5. Google+ . . . 18
2.5. Verwandte Arbeiten und Projekte . . . 18
2.5.1. What happens when Facebook is gone? . . . 18
2.5.2. Reclaim Social . . . 19

3. Analyse **21**
3.1. Ein Beispiel . . . 21
3.1.1. Beiträge lesen und konvertieren . . . 21
3.1.2. Beiträge in eine andere Webseite schreiben . . . 23
3.2. Identifizierung der Komponenten . . . 24
3.3. Wahl vorhandener Techniken . . . 24
3.3.1. Welches Zwischenformat? . . . 24
3.3.2. Welches Nachrichtensystem? . . . 24

4. Eigener Ansatz: Social Online Community Connectors (SOCC) **27**
4.1. Datenformat . . . 28
4.2. Konfiguration . . . 29
4.2.1. SOCC Connector Config Ontologie . . . 29
4.2.2. Services . . . 30
4.2.3. Benutzerdaten . . . 31
4.2.4. Authentifizierung . . . 32
4.2.5. Autorisierung . . . 34

4.3. Design eines Connectors 35
4.3.1. SOCC Context 35
4.3.2. AccessControl 36
4.3.3. ClientManager 36
4.3.4. StructureReader 37
4.3.5. PostReader 38
4.3.6. PostWriter 38
4.4. SOCC-Camel 39
4.4.1. SoccPostPollingConsumer 39
4.4.2. SoccPostProducer 40

5. Implementierung und Evaluation **43**
5.1. Verwendete Bibliotheken 43
5.1.1. RDF2Go 43
5.2. Implementierung der Connectoren 43
5.2.1. Moodle 43
5.2.2. Facebook 44
5.2.3. Google+ 45
5.2.4. Youtube 45
5.2.5. Canvas 46
5.3. Evaluation 49

6. Abschlussbetrachtung **51**
6.1. Fazit 51
6.2. Ausblick 51

A. Anhang **53**
A.1. SOCC Connector Config Ontologie 53
A.2. SIOC Services Authentication Module 55

Literaturverzeichnis **60**

Abbildungsverzeichnis

2.1. Einfacher RDF-Graph 7
2.2. Aufbau von SIOC (modifiziert) - Originalquelle: [14] 11
2.3. JMS Programmiermodell - Original Bild: `http://docs.oracle.com/javaee/1.3/jms/tutorial/1_3_1-fcs/doc/images/Fig3.1.gif` (Zugriff: 2013-09-15) . . 12
2.4. Moodle Instanz der TU Darmstadt 16
2.5. Instructure Canvas 17

3.1. Benutzer erstellt einen Beitrag im sozialen Netzwerk A. 21
3.2. Komplexität ohne und mit dem Einsatz eines Zwischenformats - Originalbild: [36] 22
3.3. Lesen des erstellten Beitrags und konvertieren in das Zwischenformat. 23
3.4. Konvertierten des Beitrags in das Format B und schreiben n das soziale Netzwerk B 23

4.1. Übersicht der Komponenten der SOCC 28
4.2. Schema der SOCC Connector Config Ontology 29
4.3. SIOC Services Module 31
4.4. Zusammenhang von Person, UserAccount und Service. Die inversen Eigenschaften `sioc:account_of` und `siocs:service_of` wurden zu einer besseren Übersicht weggelassen 32
4.5. SIOC Services Authentication Ontology 33
4.6. Basic Access Control Ontologie 34
4.8. SOCC Context 35
4.9. AccessControl 36
4.10. ClientManager 36
4.11. StructurReader 37
4.12. PostReader 38
4.7. UML Klassendiagramm eines Connectors 41
4.13. UML Sequenzdiagramm eines PostWriters 42
4.14. Übersicht des Socc-Camel Moduls in EIP-Notation 42

5.1. UML Klassendiagramm von CanvasLMS4J 50

Tabellenverzeichnis

2.1. Wichtigsten HTTP Operationen mit REST 4
2.2. Einige Beispiel von EIP 14

3.1. Anzahl Konverter bei drei Webseiten 22

4.1. Variablennamen und Ersetzung innerhalb von `unknownMessageTemplate` 30

5.1. Allgemeine Platzhalter und deren Beschreibung für das Mapping nach SIOC 43
5.2. Format der URIs für Canvas 46

Listings

2.1. SOAP Nachricht 4
2.2. RDF/XML Beispiel 7
2.3. Turtle Beispiel 8
2.4. Turtle Präfixe 8
2.5. Turtle abkürzende Schreibweise 8
2.6. FOAF Beispiel 10
2.7. JMS Beispiel 13
2.8. Apache Camel Beispiel 14

4.1. ACL Beispiel 35
4.2. SOCC-Camel Konfigurations URI 39

5.1. Canvas Authorization Header 47
5.2. CanvasLMS4J Beispielprogramm 48

1 Einleitung

Durch die Omnipräsenz des Internets im heutigen Alltag haben sich viele Bereichen unseres Lebens sehr verändert. Unter anderem die Art wie wir uns weiterbilden und neue Dinge lernen verlagert sich immer mehr dort hin. Prägend für diesen Umbruch ist der Begriff des „E-Learnings“. Besonders neue Technologien im Zuge des so genanten Web 2.0 wie Blogs, Wikis Diskussionsseiten und sozialer Netzwerke machen es immer leichter neues Wissen zu erwerben und es mit anderen zu teilen. Gerade beim Lernen spielt die Diskussion mit Gleichgesinnten eine wichtige Rolle [15]. Es wurden Studien durchgeführt, die zeigen dass Studenten welche sich an online Diskussionen teilnehmen dazu tendieren gute Noten zu bekommen, als solche die nicht teilnahmen [13, 31]. Auch für zurückhaltende Studenten ist E-Learning eine Verbesserung, da sie sich so eher an Diskussion beteiligen als zum Beispiel in der Vorlesung oder der Lerngruppe [25].

Qiyun Wang et. al. [37] zeigen in ihrer Studie, dass sich Gruppen im sozialen Netzwerk Facebook für den E-Learning Einsatz als Learning Management System (LMS) gut nutzen lassen. Teilnehmer konnte auf der Gruppenseite durch Kommentare und Chats mit einander diskutieren. Gerade die Organisation der Lerngruppe als auch die Benachrichtigung über Ereignisse funktionierte reibungslos. Jedoch uneingeschränkt konnte Facebook als LMS nicht empfohlen werden. Bemängelt wurde unter anderem die aufwändige Integration von Lernmaterialien „tutor noticed that it was quite troublesome to add teaching materials“[37, S. 435] und dass es nicht möglich war Diskussionen in einzelne Themen zu unterteilen. Alle Kommentare wurden nur als eine chronologische Liste dargestellt. Seit 2013 ist es aber auch auf Facebook möglich auf Kommentare direkt zu antworten[1] und so sind auch forumsähnliche Diskussionen realisierbar. Jedoch ist diese Erweiterung auf „Pages“ beschränkt. Über eine Ausweitung auf Gruppenseiten ist nichts bekannt.

Aber nicht nur soziale Netzwerke sind für Diskussionen innerhalb von E-Learning geeignet, Foren oder Blogs sind ebenfalls sehr beliebte Plattformen.Jedoch ein Problem bei der Nutzung des Internets zum Lernen liegt darin, dass es in der Regel nicht nur eine Plattform genutzt wird, sondern häufig mehrere simultan. Zum Beispiel könnte für einen Kurs ein eigenes Forum im LMS des Veranstalters und nebenbei noch eine Gruppe in Facebook existieren. Teilnehmer, die vorzugsweise nur eine eine der Plattformen nutzen, erhalten vielleicht von wichtigen Diskussion auf der anderen Plattform keine Kenntnis. Durch diese Inselbildung werden Themen mehrfach behandelt, da Suchfunktionen nur innerhalb der eigenen Plattform suchen und von der Existenz in der Anderen nichts wissen. Eine Integration von zusätzlichen Wissensquellen ist nur schwer möglich und erfolgt immer wieder nur in verbaler Form wie „Schau dir auf der Seite x den Artikel y an“.

Aus diesen Gründen soll in dieser Arbeit ein Ansatz entwickelt werden der er es ermöglicht verteilte Diskussionen zusammen zu führen und wiederverwenden zu können. Ein solcher Ansatz muss dazu mehrere Anforderungen erfüllen. Da es sich bei online Plattformen in der Regel um

[1] https://www.facebook.com/notes/facebook-journalists/improving-conversations-on-facebook-with-replies/578890718789613

abgeschottete „Datensilos“[5] handelt auf die nur über zum Großteil heterogene Schnittstellen zugegriffen werden kann, ist es hier wichtig eine einheitlich Schnittstelle für den Zugriff auf die gespeicherten Diskussionsdaten zu schaffen. Nicht nur in der Art des Zugriffs unterscheiden sich die einzelnen Plattformen, auch das Format der Daten ist davon Betroffen. Um Diskussionen zwischen den Plattformen überhaupt austauschen zu können ist demzufolge eine Umwandlung in ein gemeinsames Datenformat notwendig, welches erst eine Interoperabilität möglich macht. Als letztes muss noch ein System zur automatischen Synchronisation entwickelt werden wodurch verteilte Diskussionen aktuell gehalten werden können.

Diese Arbeit gliedert sich dazu in folgende Kapitel:

Kapitelbeschreibung

2 Grundlagen

mehr QUELLEN!!!

Grundlagen Einleitung schreiben

2.1 Zugriff auf Webanwendungen

Der Zugriff auf Daten von einer Webanwendung ist in den seltensten Fällen durch eine direkte Anbindung an die dahinter liegende Datenbank möglich beziehungsweise gewünscht. Gerade wenn das eigene Geschäft von diesen Daten abhängt, will man nur ungern alles mit allen teilen. Um trotzdem Dritten die Nutzung zu ermöglichen, wird dazu eine eine der Zugriff über eine vordefinierte Programmierschnittstelle (API) gestattet. Für Anwendungen und Dienste im Web sind die folgenden zwei Ansätze für die Architektur einer solchen API besonders verbreitet.

2.1.1 Representational State Transfer (REST)

Eine sehr beliebte Architektur für den öffentlichen Zugriff auf Webanwendungen ist *Representational State Transfer* (REST) [18, S. 76]. REST baut auf HTTP auf und definiert einige Beschränkungen die eine REST basierter Dienst erfüllen muss.

Die Grundidee besteht darin, dass hinter einer URL eine bestimmte Ressource sich verbirgt auf die man von außen zugreifen möchte. REST schreib aber nicht vor in welchen Datenformat diese Ressource übermittelt werden soll, sondern dass das zurückgelieferte Format der Ressource änderbar ist.

> *„REST components communicate by transferring a representation of a resource in a format matching one of an evolving set of standard data types, selected dynamically based on the capabilities or desires of the recipient and the nature of the resource.“[18, S. 87]*

Dadurch soll einer einfachere Verwendbarkeit in unterschiedliche Systemen ermöglicht werden. So kann für eine Webanwendung beim aufrufen im Browser eine HTML-Datei zurück geliefert werden, die sofort betrachtet werden kann und falls ein Programm darauf zugreift wird ein maschinenlesbares Format verwendet. Neben HTML sind auch noch XML und JSON sehr verbreitete Formate. Die Kommunikation wird dabei komplett Zustandslos abgehalten und alle Zusatzinformationen müssen immer mitgeliefert werden. Durch die Zustandslosigkeit skaliert das System viel besser, da Ressourcen sofort wieder frei gegeben werden können und nicht für spätere Anfragen gespeichert werden müssen.

Wie schon beschrieben, nutzen REST basierte Dienste HTTP als Grundlage zur Kommunikation. Die dort definierten Operationen werden mit REST zur Auslieferung und Manipulation der

Tabelle 2.1.: Wichtigsten HTTP Operationen mit REST

Operation	Beschreibung
GET	Liefert die hinter einer URL liegende Ressource an den Aufrufer zurück.
POST	Dient zum Anlegen einer neuen Ressource. Die URI der neuen Ressource ist beim Aufruf noch unbekannt und wird von Service bestimmt.
PUT	Wird zum Ändern eine bestehenden Ressource genutzt.
DELETE	löscht, wie der Name schon sagt, eine Ressource dauerhaft.

Ressourcen verwendet. Zur Grundausstattung gehören dabei GET, POST, PUT und DELETE (siehe Tabelle 2.1). Die anderen Operationen HEAD, TRACE, OPTIONS und CONNECT sind eher selten anzutreffen.

2.1.2 Simple Object Access Protocol (SOAP)

Das *Simple Object Access Protocol*[30] (SOAP) ist ein vom W3C standardisiertes Netzwerkprotokoll für den Austausch von Daten zwischen heterogenen Systemen. SOAP schreibt einen bestimmten Aufbau von Nachrichten vor, innerhalb von denen die Daten transportiert werden. Als Repräsentation für diese Nachrichten wird auf XML gesetzt. Bei der Wahl des Transportprotokolls werden dahingegen keine Vorgaben gemacht und es ist frei wählbar. Häufig wird es aber in Verbindung mit HTTP und TCP verwendet.

Listing 2.1: SOAP Nachricht

```
1 <Envelope xmlns="http://www.w3.org/2003/05/soap-envelope">
2     <Header>
3         <!-- header information -->
4     <Header>
5     <Body>
6         <!--body content-->
7     </Body>
8 </Envelope>
```

Eine Nachricht besteht im Grunde aus drei Elementen: den *Envelope*, einen optionalen *Header* und einem *Body* (siehe Listing 2.1). Der Envelope fungiert, wie die Übersetzung schon sagt, als Briefumschlag für die zu transportierenden Daten. Innerhalb jedes Envelopes können zusätzliche Meta-Informationen im Header Element gespeichert werden. Die eigentlichen Daten befinden sich im Body Element des Evelopes. Wie der Inhalt von Header und Body auszusehen haben wird von SOAP nicht vorgeschrieben. Dies können weitere XML Elemente oder einfache Zeichenketten sein.

Web Services Description Language

Gebräuchlich ist der Einsatz von SOAP bei sogenannten *Remote Procedure Calls* (RPC). Unter RPC verseht man den Aufruf eine Funktion von einem entfernten Dienst und das Zurückliefern einer eventuell vorhandenen Antwort. Welche Funktionen von einen Dienst zur Verfügung stehen wird ein einer *Web Services Description Language*[12] (WSDL) Datei beschrieben. Diese WSDL Datei wird in XML Format geschrieben und enthält alle wichtigen Informationen für RPC Aufrufe, die von einen Dienst zur Verfügung gestellt werden:

types enthält Definition von Datentypen die in einer Message eingesetzt werden können. Zur Definition der Datentypen wird das Vokabular von XML Schema[1] eingesetzt.

message Elemente beschreiben die Datentypen aus denen eine Nachricht aufgebaut ist.

portType definiert eine Menge an zur Verfügung stehenden Operationen. Inklusive Eingabe- und Ausgabeparameter. In der WSDL Version 2.0 wurde portType in *interface* umbenannt.

binding beschreibt das Format und den Protokollablauf mehrerer Operationen. Zum Beispiel wie Eingabe- und Ausgabeparameter kodiert werden sollen.

port Definiert eine Adresse hinter der sich ein Binding befindet. Üblicherweise in Form ein URI. Seit WSDL 2.0 wird statt port der Begriff *endpoint* verwendet.

service dient zum Zusammenfassen mehrerer Ports zu einen einzigen Dienst.

Wird eine solche WSDL Datei öffentlich zugänglich gemacht, kann festgestellt werden welche Funktionen ein Dienst anbietet und automatisch APIs für unterschiedliche Systeme generiert werden. Der weiter Datenaustausch erfolgt dann über SOAP Nachrichten.

2.2 Datenintegration

Datenintegration-Einleitung schreiben

2.2.1 Semantic Web

Seit den Anfängen des *World Wide Webs* (kurz als WWW oder Web bezeichnet) hat die Masse an abrufbaren Information immer mehr zugenommen. Die Vorteile des Webs liegen eindeutig in der guten Aktualität und Erreichbarkeit von überall auf Erde. Die Menge an Informationen sind aber auch ein Problem im Web. Da diese überall verteilt sind, ist es schwer für einen einzelnen alleine alles zu einem Thema zu finden. Suchmaschinen wie Google[2], Yahoo[3] oder Microsoft Bing[4] leisten hier gute Dienste. Doch für Maschinen ist es noch immer nicht einfach die Inhalte von Webseiten zu verstehen, da diese für Menschen gemacht wurden [22]. Auch der erlernen

1 `http://www.w3.org/XML/Schema`
2 `https://www.google.com`
3 `yahoo.com`
4 `http://www.bing.com/`

von neuen Wissen anhand vorhandener Informationen ist in der aktuellen Form des Webs nur schwer realisierbar.

2001 machte Tim Berners-Lee (der Erfinder des Webs) den Vorschlag [6] das Web mit maschinenlesbaren Informationen zu erweitern und so die Verarbeitung mit Computerprogrammen zu vereinfachen. Die Idee des *Semantic Webs* wurde geboren. Der Inhalt des Webs wird mit semantischen Information so erweitert, dass Programme das sich zwei Texte an unterschiedlichen Stellen des Webs um das selbe Thema handeln. Aber auch die Anzeige von impliziten Wissen, wenn jemand zum Beispiel eine Telefonnummer in Los Angeles, USA sucht und aus Deutschland anrufen will, wird im mitgeteilt dass er wegen der Zeitdifferenz von neun Stunden lieber erst Nachmittags anrufen sollte, wäre so einfacher möglich.

Resource Description Framework

Eine Baustein des Semantic Webs ist das *Resource Description Framework* (RDF). Wie der Name schon suggeriert, dient RDF zur Beschreibung von einzelnen Ressourcen innerhalb des Webs. Nach [27, 28] 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“[27].

Das Datenmodell von RDF ist zur effizienten Verarbeitung sehr einfach aufgebaut. Die Grundlage bilden Tripel aus Subjekt, Prädikat und Objekt. Einer oder mehrere solcher Triple zusammen werden als gerichteter RDF-Graph bezeichnet. Subjekt und Objekt stehen über das Prädikat mit einander in Beziehung, wobei die Beziehung immer vom Subjekt zum Objekt geht. Das Prädikat wird auch als Eigenschaft (engl. Property) bezeichnet. Gemeinsam beschreibt das Triple immer eine Aussage über eine oder zwei Ressourcen. Zum Beispiel „Die Dose enthält Kekse“ wäre eine Aussage, dass in einer Dose sich Kekse befinden. Die Dose ist dabei das Subjekt, enthält das Prädikat und Kekse das Objekt. Ein Triple ist quasi ein einfacher Satz in der natürlichen Sprache [21]. Für Subjekt, Prädikat und Objekt werden in RDF *Uniform Resource Identifier* (URI), *Literale* oder *leere Knoten* (im englischen *Blank Nodes* genannt) verwendet.

URIs sind eindeutige Bezeichner die eine beliebige reale oder abstrakte Ressource darstellen und werden wie in RFC 2396[5] beschrieben formatiert. Relative URIs sollten aber nach [27] aber nach Möglichkeit vermieden werden. URIs bilden eine Verallgemeinerung der im Web gebräuchlichen Uniform Resource Locator (URL).

Literale bestehen aus einfachen Zeichenketten die zum Speichern der Informationen dienen. Zusätzlich können Literale mit der Angabe der verwendeten Sprache `Öbjekt"@de` oder des Datentyps `"42"8sd:integer` erweitert werden. Bei Literalen ist darauf zu achten, dass die Literale `Öbjekt"` und `Öbjekt"@de` auf den ersten Blick zwar den selben Wert beschreiben, aber aus Sicht von RDF nicht die selben sind. Sowohl die angegebene Spracht als auch der Datentyp müssen übereinstimmen.

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

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 nicht unbedingt eine eigene URI nötig ist und sind nur innerhalb eines Graphen eindeutig. Für die Referenzierung außerhalb des RDF-Graphen sind leere Knoten ungeeignet.

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.

Darstellung von RDF-Graphen

In Laufe der Zeit von RDF wurde verschiedene Möglichkeiten erfunden einen RDF-Graphen darzustellen. In diesem Abschnitt werden drei Formen vorgestellt die auch in dieser Arbeit zur Visualisierung benutzt werden.

Graphische Darstellung

Graphisch lässt sich RDF als gerichteter Graph mit Knoten und Kanten darstellen. Ressourcen werden dabei als elliptische Knoten, Literale als Rechtecke und die Prädikate als gerichtete Kante gezeichnet. Ein Beispiel ist in Abbildung 2.1 zu sehen.

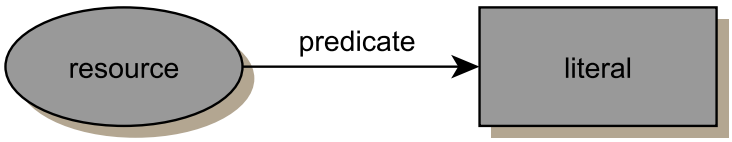


Abbildung 2.1.: Einfacher RDF-Graph

RDF/XML

RDF/XML[28, Abschnitt 3.2] ist eine verbreitete Form RDF-Dokumente zu beschreiben. Die Basis bildet hierbei die Verwendung der Extensible Markup Language (XML). In Listing 2.2 ist ein Beispieldokument in RDF/XML zusehen. Das in Zeile 2 zu sehende `rdf:RDF` Element zeigt, dass sich innerhalb von ihm sich die RDF-Beschreibung des Dokuments befindet. In diesem ELement werden mit `xmlns:` einige Präfixe für Namensräume definiert, um das Dokument übersichtlicher zu halten. Alle Präfixe werden danach mit den angegebenen Namensraum ersetzt. Das `Description` in Zeile 5 stellt die Beschreibung einer Ressource im RDF-Graphen. Die URI der Ressource wird mit dem Attribut `rdf:about` definiert. Innerhalb des Description Elements befinden sich die Prädikate. In Zeile 6 steht also, dass die Ressource die Eigenschaft `exterms:enthaelt` besitzt und diese das Literal `Kekse`. Wäre das Objekt nicht wie hier ein Literal sondern eine weitere Ressource, könnte man über das Attribut `rdf:ressource` für das `exterms:enthaelt` auf diese Ressource verweisen.

Listing 2.2: RDF/XML Beispiel

```
1 <?xml version="1.0"?>
```

```
2 <rdf:RDF xmlns:rdf="http://www.w3.org/1999/02/22-rdf-syntax-ns#"
3     xmlns:exterms="http://www.example.org/terms#">
4
5     <rdf:Description rdf:about="http://www.example.org/dose">
6        <exterms:enthaelt>Kekse</exterms:enthaelt>
7     </rdf:Description>
8 </rdf:RDF>
```

Turtle

Turtle (Ausgeschrieben: *Terse RDF Triple Language*) ist eine weiter Möglichkeit RDF-Graphen darzustellen und ist eine ging aus der Sprache N3 (Kurzform für Notation 2) hervor [3]. In Turtle wird das Triple aus Subjekt, Prädikat und Objekt hintereinander geschrieben und zwischen jeden mindestens ein Leerzeichen gelassen. Als Abschluss folgt nach jedem Tripel noch ein Punkt. Der Punkt verdeutlicht noch einmal die Ähnlichkeit mit gesprochenen Sätzen. Listing 2.3 zeigt das Beispiel mit der Keksdose noch einmal in Turtle Notation.

Listing 2.3: Turtle Beispiel

```
1 <http://www.example.org/dose> <http://www.example.org/terms#enthaelt>
      "Kekse" .
```

In Turtle ist darauf zu achten, dass alle URIs immer zwischen Spitzenklammern stehen müssen. Literale werden in Anführungszeichen geschrieben. Da nun einzelne Prädikate beziehungsweise allgemein URIs recht häufig innerhalb eines Graphen auftauchen können, kann es einfacher sein diese abzukürzen. Wie schon in RDF/XML können auch in Turtle Präfixe definiert werden um Schreibarbeit zu sparen.

Listing 2.4: Turtle Präfixe

```
1 @prefix exterms: <http://www.example.org/terms#> .
2 <http://www.example.org/dose> exterms:enthaelt "Kekse" .
```

In der ersten Zeile von Listing 2.4 wird durch Einleiten mit dem Schlüsselwort `@prefix` ein neuer Präfix `exterms:` für den Namensraum `http://www.example.org/terms#` festgelegt. Dieser Präfix kann nun überall innerhalb des Dokumentes verwendet werden, wobei die Spitzenklammern dann weggelassen werden können.

Listing 2.5: Turtle abkürzende Schreibweise

```
1 @prefix exterms: <http://www.example.org/terms#> .
2 <http://www.example.org/dose> exterms:farbe "blau";
3     exterms:enthaelt "Kekse", "Geld" .
```

Listing 2.5 zeigt nochmal ein drittes Beispiel, wie redundante Angeben eingespart werden. Wie man in der zweiten Zeile sehen kann. wir deine Frage für die Dose angeben das Triple aber mit einen Semikolon abschlossen und nicht mit einen Punk. Durch das Semikolon ist es Möglich das Subjekt mehrfach wieder zu verwenden, wenn sich nur Prädikat und Objekt ändern. So können sich mehrere Eigenschaften einer Ressource platzsparend schreiben ohne das Subjekt immer

wieder anzugeben. Ändert sich dahingegen nur das Objekt können mehrere durch Kommata getrennt hintereinander geschrieben werden. Die dritte Zeile beschreibt zum Beispiel, dass in der Dose nicht nur Kekse sonder auch Geld steckt. Leere Knoten können dann noch in Turtle durch angeben einer geöffneten eckigen Klammer gefolgt von einer sich Schließenden dargestellt „`[]`“. Soll ein leerer Knoten innerhalb eines Graphen referenziert werden, kann er auch als `_:LABEL`, wobei `LABEL` ein beliebiger Beizeichner ist, geschrieben werden.

2.2.2 Ontologien

Sollen Daten aus verschiedenen Quellen zusammengefügt werden, stellt sich häufig das Problem dass Teile dieser dieser Daten zwar den gleichen Sinn haben, aber aufgrund der Sichtweise des jeweiligen System eine andere Bezeichnung besitzen. Das kann zum Beispiel zu Missverständnissen bei der Verarbeitung führen oder dass Teile eines anderen Systems nicht wiederverwendet werden können [36]. Einen Ausweg aus diesem Dilemma kann die Verwendung von Ontologien zeigen. Ontolgien können allgemein als Wissensbasis [36, 22] bezeichnet werden und liefern eine formale Spezifikation über eine bestimmte Interessensdömäne. Sie beschreibt nicht nur wie das verwendete Vokabular aussieht, sondern legt auch fest welche einheitliche Bedeutung jede Vokabel hat.

Im Bereich des Semantic Webs sind heutzutage zwei Sprachen für die Erstellung von Ontologien weit verbreitet. Diese sind *RDF Schema* (RDFS)[10] und die darauf aufbauende *Web Ontology Language* (OWL)[32]. Beide Sprachen basieren auf RDF, so können sie zusammen mit jedem System verwendet werden, das RDF versteht. Mit ihnen ist man im Stande Klassen von abstrakten Objekten und deren Eigenschaften zu definieren und diese in eine Vererbungshierarchie einzugliedern. Im Gegensatz zu RDFS können in OWL diverse Einschränkungen definiert werden, wie zum Beispiel dass eine Eigenschaft nur einmal pro Objekt einer Klasse vorhanden sein darf. Die Anhänge A.1 und A.2 zeigen zwei Ontologien in der Sprache OWL, welche innerhalb dieser Arbeit entwickelt wurden und in Abschnitt 4.2.1 und 4.2.4 beschreiben werden.

überleitung zu FOAF und SIOC

Friend of a Friend (FOAF)

Friend of a Friend[6] (FOAF) ist ein 2000 gestartet Projekt und versucht Personen innerhalb des Webs, inklusive der Verbindungen zwischen ihnen und anderen, sowie dem was sie machen, in maschinenlesbarer Form abzubilden. FOAF stellt hierzu ein Vokabular [11] auf der Basis von RDF für solche sozialen Netzwerke zur Verfügung. Das Vokabular von FOAF gliedert sich dazu in einen „FOAF Core“ und einen „Social Web“ Bereich. Der Core-Bereich die Klasse `Agent` für alle Dinge die eine Handlung ausführen können, also sowohl natürliche Personen, Gruppen oder Organisationen als auch Computerprogramme oder Maschinen. Für diese gibt es jeweils noch einzelne Unterklassen die von der Klasse `Agent` erben. Objekten dieser Klassen können eigene Eigenschaften wie einen Namen, ein Alter oder wen sie kennen gegeben werden. Der Sozial Web Bereich enthält alle Teile die für das Web interessant wären. Das wären zum Beispiel welche E-Mail-Adresse eine Person besitzt, welche Benutzerkonten im bei welcher Webseite gehören

[6] `http://www.foaf-project.org`

oder wie die URL seiner Homepage lautet. Das FOAF Projekt sieht sich aber selber nicht als Konkurrenz gegenüber den etablierten sozialen Online-Netzwerken, sondern eher ein Ansatz für einen besser Austausch zwischen den einzelnen Seiten [11, Abstract].

Listing 2.6 zeig eine Beispiel FOAF-Dokument in RDF/XML. Es beschreibt die fiktive Person „Max Mustermann“, mit Vor- und Nachname sowie der Hashwert seiner E-Email-Adresse (`foaf:mbox_sha1sum`). Diese Person kennt (`foaf:knows`) kennt eine Person mit dem Namen „John Doe“ der ebenso eine E-Mail-Adresse mit den angegeben Hashwert besitzt. Statt die Eigenschaften von John Doe hier noch einmal vollständig anzugeben, wird über die Eigenschaft `rdfs:seeAlso` auf ein weiteres FOAF-Dokument verwiesen, dass die fehlenden Daten enthält.

Listing 2.6: FOAF Beispiel

```
1  <rdf:RDF xmlns:rdf="http://www.w3.org/1999/02/22-rdf-syntax-ns#"
2           xmlns:rdfs="http://www.w3.org/2000/01/rdf-schema#"
3           xmlns:foaf="http://xmlns.com/foaf/0.1/">
4    <foaf:Person>
5      <foaf:name>Max Mustermann</foaf:name>
6      <foaf:firstName>Max</foaf:firstName>
7      <foaf:surname>Mustermann</foaf:surname>
8      <foaf:mbox_sha1sum>dce4fc922158f8b26fbf0a65ea32bfab58488bd2</
       foaf:mbox_sha1sum>
9      <foaf:knows>
10       <foaf:Person>
11         <foaf:name>John Doe</foaf:name>
12         <foaf:mbox_sha1sum>479ea35d3522662b70dc7afd721853485c95db57</
       foaf:mbox_sha1sum>
13         <rdfs:seeAlso rdf:resource="http://www.example.org/people/jd/
       foaf.rdf"/>
14       </foaf:Person>
15     </foaf:knows>
16   </foaf:Person>
17 </rdf:RDF>
```

Semantically-Interlinked Online Communities (SIOC)

Semantically-Interlinked Online Communities (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 [14, 9, 8]. 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 FOAF und für einige Inhaltliche- und Metadaten (Titel, Inhalt, Erstelldatum, ...) Dublin Core Terms[7] eingesetzt.

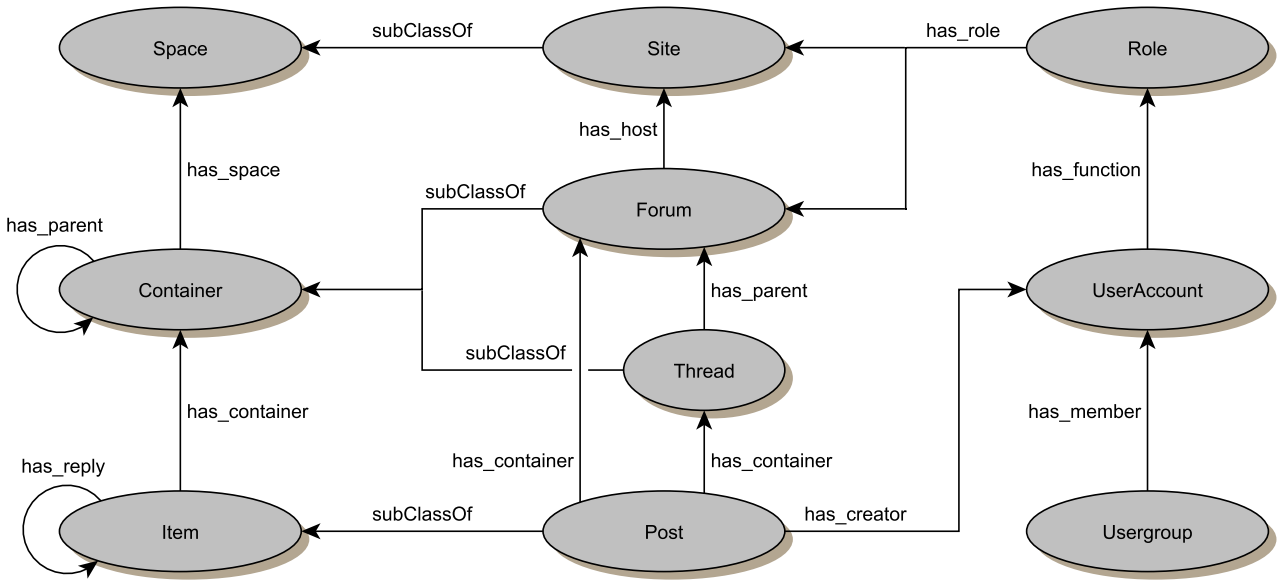


Abbildung 2.2.: Aufbau von SIOC (modifiziert) - Originalquelle: [14]

Die wichtigsten Klassen von SIOC sind in Abbildung 2.2 in der mittleren Spalte zu sehen. Die Klasse `Site` ist für die Beschreibung von allgemeinen Webseiten in denen Beiträge innerhalb von Containern verfasst werden. Ein solcher Container ist die Klasse `Forum`und steht für einen Ort an dem Diskussionen geführt werden. Enthält ein Forum unterschiedliche Diskussionen zu unterschiedlichen Themen, kann es noch einmal in unterschiedliche `Thread` unterteilt werden, welche immer ein Forum als Elternteil haben (`has_parent`). Beide Klassen leiten sich von der Klasse `Container` als einen allgemeinen Ort für Beiträge ab. Die einzelnen Beiträge werden durch die Klasse `Post` beziehungsweise von der übergeordneten Klasse `Item` modelliert. Beiträge gehören in der Regel immer zu einen bestimmten Container oder mindesten zu einer Webseite. Es ist auch mögliche Beiträge als Kommentar zu anderen Beiträgen über die Eigenschaft `has_reply` abzubilden. Jeder Beitrag besitzt mindesten einen Autor der ein Benutzerkonto auf der betreffenden Seite besitzt. Für die Beschreibung eines solchen Benutzerkontos wird die Klasse `UserAccount` verwendet. Dieses Benutzerkonto kann nun zu einer Gruppe von anderen Konten gehören, zum Beispiel einer Lerngruppe. Über die Klasse `Role` keine einen Benutzerkonto eine bestimmte Rolle innerhalb eine Seite, Forum und so weiter zugeteilt werden. Ein Beispiel dafür wäre die Rolle eines Moderator, der überwacht ob die Regel der Seite in den einzelnen Foren eingehalten werden.

2.3 Datenverteilung

Datenverteilung-Einleitung schreiben

[7] `http://dublincore.org/documents/dcmi-terms`

2.3.1 Java Messaging Service

Das *Java Messageing Service* (JMS) ist ein Sammlung von Schnittstellen für das Erstellen, Senden und Empfangen von Nachrichten zwischen Clients [34]. JMS erlaubt eine Entwicklung von verteilten Anwendung die nicht nur lose gekoppelt, sondern auch asynchron und zuverlässig arbeiten.

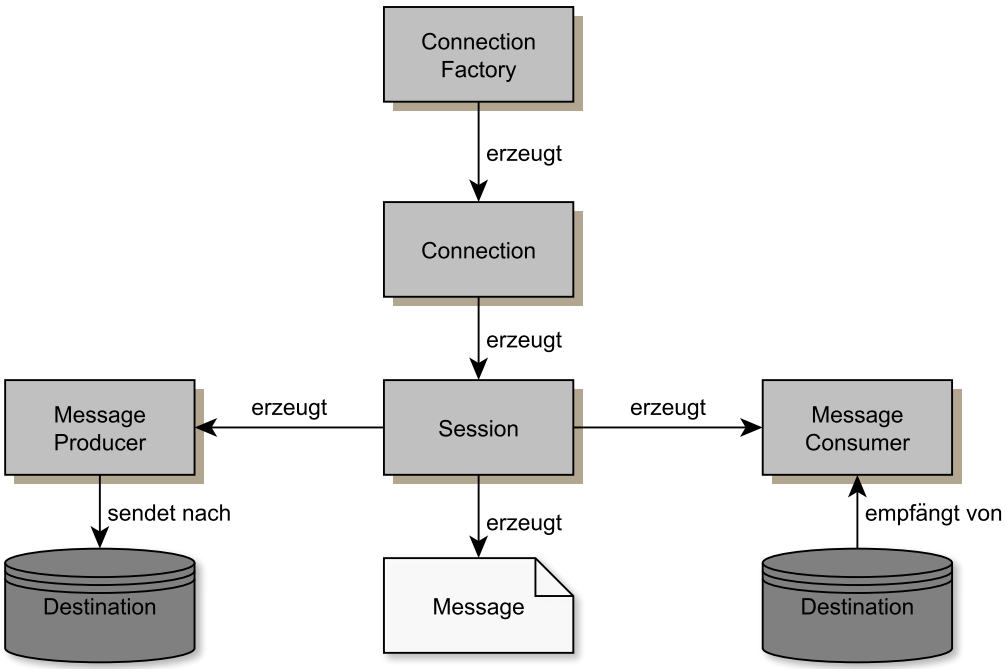


Abbildung 2.3.: JMS Programmiermodell - Original Bild: `http://docs.oracle.com/javaee/1.3/jms/tutorial/1_3_1-fcs/doc/images/Fig3.1.gif` (Zugriff: 2013-09-15)

Die Grundstruktur einer JMS Anwendung besteht aus einem *JMS Provider*, der die Schnittstellen von JMS implementiert. Dieser JMS Provider wird auch all *Message Oriented Middleware* (MOM) bezeichnet und kümmert sich auch darum, dass Nachrichten zuverlässig verschickt werden. Einen JMS Client von dem Nachrich ten verschickt und empfangen werden. Den Nachrichten selber und den sogenannten *Administered Objects*. Diese bestehen aus vorkonfigurierten *Connection Factorys* für das Erstellen von Verbindungen (Connections) zwischen Client und MOM und *Destinations* als Sende- und Empfangspunkte von Nachrichten. Die Administered Objects können im Client über das Java Naming and Directory Interface (JNDI)[8] API abgefragt werden. Alle Clients die nicht die JMS API sondern die implementierte API der MOM direkt verwenden, werden *Native Client* genannt.

JMS unterstützt zwei Verbindungsarten zum Übertragen von Nachrichten: Queue-basiert und Topic-basiert. Als Queue-basiert wird eine Punkt-zu-Punkt Verbindung bezeichnet. Hier werden Nachrichten nur zwischen zwei Clients übertragen und gegebenenfalls in einer Warteschlange zwischengespeichert. Hinter Topic-basiert verbirgt sich ein Publish-Subscribe Mechanismus bei dem ein Client Nachrichten an eine bestimmtes Topic-Destination schickt und andere Clients

[8] JNDI API:`http://www.oracle.com/technetwork/java/index-jsp-137536.html`

sich auf dieses Topic anmelden können, die dann die Nachrichten des ersten Clients zugeschickt bekommen. Ob Queue oder Topic-basiert, wird über das verwendete Destination Objekt ausgewählt.

Sollen nun Nachrichten von einem Client verschickt beziehungsweise empfangen werden, muss mit einer Connection Factory eine neue Connection zu einer MOM aufgebaut werden. Mit dieser Connection wird danach ein *Session*-Objekt erstellt, das als Kontext zum Senden und Empfangen verwendet wird. Sollen Nachrichten gesendet werden, muss mit einem Destination-Objekt ein *MessageProducer* und danach eine Nachricht mit dem Session-Objekt erstellt werden. Die Nachricht wird dann über den MessageProducer versendet. Für das Empfangen ist ein *MessageConsumer* verantwortlich. Abbildung 2.3 zeig noch einmal den Zusammenhang aller JMS Komponenten.

Listing 2.7: JMS Beispiel

```
1  Context ctx = new InitialContext();
2  ConnectionFactory connectionFactory = (ConnectionFactory) ctx.lookup(
       "ConnectionFactory");
3
4  Connection connection = connectionFactory.createConnection();
5  connection.start();
6
7  Session session = connection.createSession();
8  Destination destination = session.createTopic("topic-test");
9
10 MessageProducer msgProducer = session.createProducer(destination);
11 Message msg = session.createTextMessage("Hallo World!");
12 msgProducer.send(msg);
```

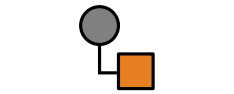
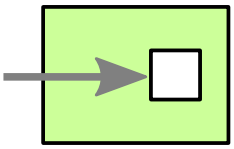
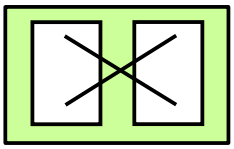
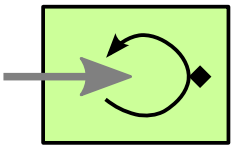
Listing 2.7 zeig ein kleines Beispiel zum Senden eine Textnachricht mit JMS. Die erste und zweite Zeile zeigt wie ein JNDI Kontext erstellt und nach einer vordefinierten Connection Factory mit den Name „ConnectionFactory“ gesucht wird. Mit dieser Connection Factory wird dann eine neue Connection erstellt und gestartet. Danach folgt in Zeile 7 und 8 das Erstellen einer neuen Session und die Definition eins Topics mit dem Namen „topic-test“ als Destination. In der zehnten Zeile wird dann der MessageProducer zum Senden von Nachrichten und in der Folgezeile die zu sendende Textnachricht erzeugt. Diese wird dann in der letzten Zeile an das Topic verschickt.

2.3.2 Enterprise Integration Pattern (EIP)

Bezeichnungen wie „Iterator“, „Factory Method“, „Observer“ oder „Proxy“ hat bestimmt schon jeder Programmieren mindestens einmal gehört. Hierbei handelt es sich im sogenannte Entwurfsmuster für Softwareprogramme. Sie sind Schablonen für Lösungen von Problemen, die in der Entwicklung von Software immer wieder auftreten und sich als hilfreich erwiesen haben. Auch in der Integration von verschiedenen Geschäftsanwendungen in ein System treten solche Muster immer wieder auf. Gregor Hohpe und Bobby Woolf beschreiben in ihren Buch „Enterprise Integration Patterns: Designing, Building, and Deploying Messaging Solutions“[23] eine Vielzahl solcher Enterprise Integration Patterns (EIP) für die Integration mit MOM. Alle hier aufzuzäh-

len würde den Rahmen dieser Arbeit sprengen. Aus diesem Grund werden in Tabelle 2.2 fünf Muster inklusive der verwendeten Symbole vorgestellt die später noch vorkommen werden. Die restlichen Muster sind auf der Webseite von EIP[9] zu finden.

Tabelle 2.2.: Einige Beispiel von EIP

Icon	Name	Beschreibung
	*Message*	Über Nachrichten werden Daten zwischen zwei oder mehr Systemen ausgetauscht.
	*Channel*	Ein Channel beschreibt einen Nachrichtenkanal über dem Nachrichten von einem Systemen in ein anderes verschickt werden können.
	*Endpoint*	Endpoints sind Schnittstellen in einem System, von dem Nachrichten in einen Kanal gesendet oder von da Empfangen werden.
	*Message Translator*	Nicht immer liegt eine Nachricht im richtigen Format für ein System vor. Durch Message Translators können diese in das gewünschte Format übersetzt werden.
	*Polling Consumer*	

Apache Camel

Apache Camel[2] (kurz: Camel) ist ein Projekt der Apache Software Foundation (kurz: Apache) für das Routen und Verteilen von Nachrichten zur Integration von System auf der Basis von definierten Regeln. Camel stellt dazu eine Java basierte API für den Einsatz der EIP bereit. Die Regeln für die Routen, die Nachrichten nehmen können, können direkt in Java aber auch durch das Spring Framework[10] in XML definiert werden.

Listing 2.8: Apache Camel Beispiel

```
1 RouteDefinition rd = new RouteDefinition()
2     .from('timer://helloTimer?period=3000')
3     .to('log:helloLog');
4
5 CamelContext camelContext = new DefaultCamelContext();
6 camelContext.addRouteDefinition( rd );
7 camelContext.start();
```

[9] http://www.enterpriseintegrationpatterns.com/toc.html
[10] http://www.springsource.org/

Listing 2.8 zeig ein kleines Beispiel, wie eine Nachrichtenroute in Java definiert werden kann. In der ersten Zeile wird über die Klasse `RouteDefinition` eine neue Route erstellt. Über die Methode `from` wird der Endpunkt vom dem die Route ausgeht festgelegt. Welcher Endpunkt das genau seien soll kann auf zwei Arten definiert werden. Man übergibt der Methode direkt ein Objekt einer Klasse die das Interface `Endpoint` implementiert oder man macht es, wie hier im Beispiel, über eine URI. Die URI baut sich auf folgende Weise zusammen. Der Teil bis zum Doppelpunkt, das sogenannte „Schema“, legt die Komponente (engl. `Component`) fest, von der ein Endpunkt erzeugen werden soll. In diesem Falls ist es eine Timer-Komponente, deren Endpunkt im periodischen Abstand Event-Nachrichten verschickt. Der Rest der URI wird zur Konfiguration an den Endpunkt übergeben. „helloTimer“ steht hier für einen Namen für den Timer und der Parameter „period“ gibt den zeitlichen Abstand zwischen zwei Event-Nachrichten an. Das Ziel der Route wird mit der Methode `to` in Zeile Drei festgelegt. Für das Ziel wird hier eine Log-Komponente mit dem Namen „helloLog“ festgelegt, die alle reinkommenden Nachrichten protokolliert. In der fünften Zeile wird ein `CamelContext` Objekt, das für die Verwaltung und Ausführung der Routen verantwortliche ist. Die eben erstellte Route wird dann dem CamelContext hinzugefügt und die Ausführung in der letzten Zeile gestartet. Nun Wird der Timer alle 3000 Millisekunden eine neue Event-Nachricht erzeugen und an den CamelContext schicken. Dieser leitet die Nachricht dann an das durch die Route definierte Ziel und wird dort von der Log-Komponente

2.4 Lernplattformen und soziale Online-Netzwerke

An dieser Stelle sollen noch kurz einige Lernplattformen und soziale Online-Netzwerke vorgestellt, die im späteren Verlauf dieser Arbeit für die Implementierung verwendet wurden. Im Einzelnen waren dies *Moodle*, *Canvas*, *Youtube*, *Facebook* und *Google+*, da sie einen guten Schnitt von den Plattformen bilden, die heutzutage sowohl im Bereich E-Learning als auch von der breiten Masse verwendet werden.

2.4.1 Moodle

Moodle[11] ist ein weit verbreitetes Open Source Online LMS. Die Hauptaufgabe liegt im Verwalten von online Kursen im Bereich E-Learning. Hierzu bietet Moodle von Haus aus eine große Menge an Funktionen für die Verwaltung des Kurses und die Kommunikation zwischen Lehrenden und Lernenden. Es bietet die Möglichkeit Aufgaben die Teilnehmern zu verteilen, Fragebögen zu erstellen, zusätzlichen Kursmaterialien bereitzustellen und den Lernerfolg durch Benotung und Feedback zu kontrollieren. Funktionen für die Unterstützung des kollaborativen Lernens sind ebenfalls vorhanden. Teilnehmer können Lerngruppen bilden, sich über persönliche Nachrichten austauschen, gemeinsam an Wikis arbeiten oder in Foren diskutieren.

Moodle wurde in der Programmiersprache PHP geschrieben und unterstützt die Datenbanken werden MySQL, PostgrSQL, MSSQL und Oracle. Die Installation von weiteren Funktionalitäten ist durch von Dritten geschriebenen Erweiterungen möglich. Seit Version 2.0 können für Moodle auch Webservices installiert werden, so können auch externe Anwendungen auf interne Funktionen und Daten zugreifen.

[11] `https://moodle.org/`

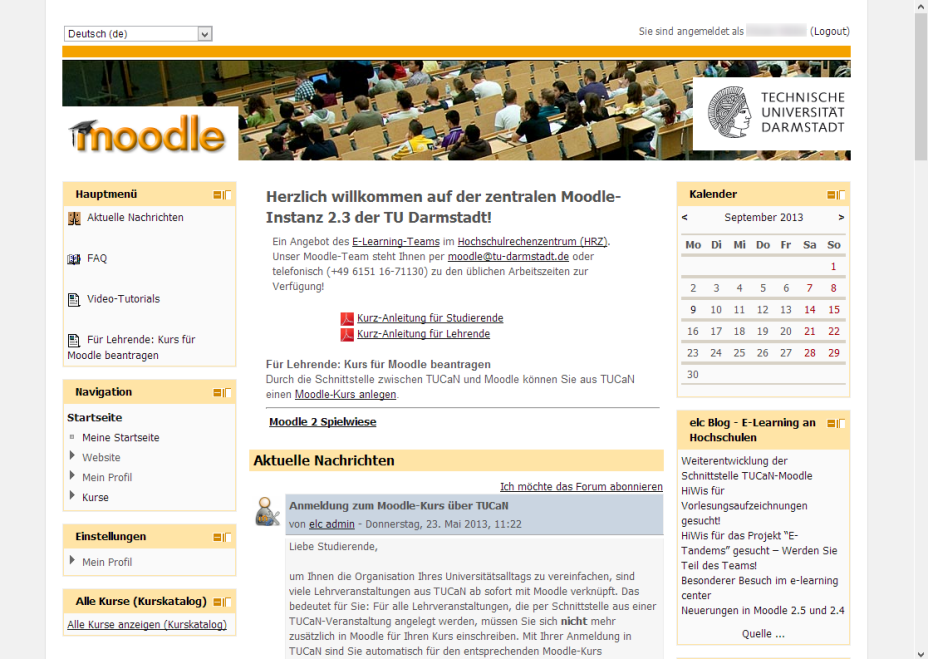


Abbildung 2.4.: Moodle Instanz der TU Darmstadt

2.4.2 Canvas

Das von der Firma Instructure[12] entwickelte *Canvas* ist ein unter Open Source Lizenz gestelltes LMS. Vom Funktionsumfang ist es Moodle nicht unähnlich. Es existiert eine Verwaltung einzelner Kurse. Innerhalb dieser Kurse können in einem Forum Diskussionen geführt und Lernmieteralien hoch- und heruntergeladenen werden. Verteilung von Aufgaben, deren Benotung und ein Benachrichtigungssystem existiert ebenfalls. Canvas erlaubt auch das Einbinden von externen Diensten zum kollaborativen Lernen und Arbeiten wie Google Docs[13] oder der Webkonferenz Anwendung BigBlueButton[14].

Canvas wird mittels des Webframeworks *Ruby on Rails*[15] entwickelt. Das Aussehen ist etwas moderner, als das von Moodle und es wird sehr stark auf die neuesten Webtechnologien wie HTML5 CSS3 und JQuery gesetzt. Eine Erweiterung der Funktionalität von Canvas ist durch das einbinden von Programmen möglich, die den *Learning Tools Interoperability*™ (LTI) Standard erfüllen. Einige solcher Programme finden sich auf der Webseite `https://www.edu-apps.org`. Unter anderem Programme zum Suchen und Einbinden von Youtube Videos, Wikipedia Artikeln, GitHub Gists[16] und vielen weiteren.

12 `https://www.instructure.com/`
13 `https://drive.google.com`
14 `http://www.bigbluebutton.org`
15 http://rubyonrails.org/
16 `https://gist.github.com`

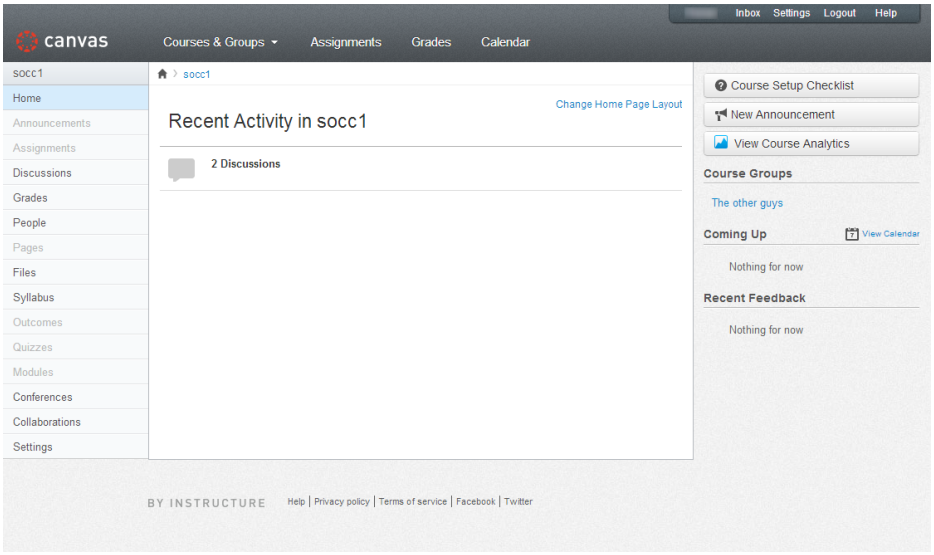


Abbildung 2.5.: Instructure Canvas

2.4.3 Youtube

Die Youtube[17] Webseite gehört wohl heute zu den beliebtesten Anlaufpunkten im Internet, wenn es um das Thema Videos geht. Monatlich nutzen über 1 Milliarde Nutzer die Seite und pro Minute werden 100 Stunden neuer Videos hochgeladen [38]. Doch nicht das komplette Videomaterial besteht aus Katzen, Musik oder Videos von Unfällen. Ein Teil der Benutzer die eigene Videos hochladen, wollen anderen Dinge beibringen, weil es sie schon immer interessierte oder früher selber Probleme damit hatten. Einer erklärt die Logarithmengesetze, ein anderer wie man Feuer ohne Feuerzeug macht und eine ganz andere gibt Schönheitstipps. Youtube ist also auch im E-Learning Bereich gut einsetzbar. Lehrende können eigene Videos hochladen, von anderen interessante Videos in Playlisten zusammenfassen und die Lernenden können über Kommentare Fragen zum Inhalt stellen.

2.4.4 Facebook

Das soziale Online-Netzwerk Facebook[18] kann mit rund 699 Millionen aktiven Benutzern täglich [16] zu den aktuell beliebtesten Vertretern seiner Art bezeichnet werden. Facebook erlaubt es, wie alle sozialen Online-Netzwerke, bekannte Personen in Freundeslisten zusammen zufassen und mit ihnen private Nachrichten auszutauschen. Beiträge wie Texte, Fotos oder Videos können auf einer Art Pinnwand der „Wall“ öffentlich oder nur mit Freunden geteilt werden. Benutzer mit gemeinsamen Interessen können dazu eigene Gruppen bilden und dort auf einer eigen Wall Beiträge veröffentlichen oder die anderer kommentieren. Wie in der Einleitung schon erklärt zeigt Qiyun Wang et. al. [37] das sich Facebook, wenn auch mit Einschränkungen, wunderbar zur Verwaltung und Nutzung durch Lernkurse und Lerngruppen eignet. Die gleiche Erfahrung teilte

[17] `https://www.youtube.com`
[18] `https://www.facebook.com/`

Anthony Fontana [19, 17], der Facebook als Alternative zum bestehenden System der Bowling Green State University in Ohio, USA verwendete.

2.4.5 Google+

Google+[19] ist ein 2011 von Google gestartetes soziales Online-Netzwerk. Seit Anfang 2013 ist Google+, von der Anzahl der aktiven Benutzer her gesehen, auf Platz 2 hinter Marktführer Facebook [35]. Vom Funktionsumfang sind sich beide sehr ähnlich. Auf Google+ können andere Benutzer in sogenannten „Circles“ sortiert werden. Dies entspricht ungefähr den auf Facebook genutzten Freundeslisten. Jeder Benutzer hat einen eigenen „Stream“ in dem er Beiträge öffentlich oder nur für ein oder mehrere Circles verfassen kann. Das Gründen von Gruppen für bestimmte Interessensbereiche ist auch in Google+ möglich und werden dort als „Communities“ bezeichnet. Eines der interessantesten Funktionen von Google+ dürfte die Einführung von „Google Hangout“ sein. Hier können Benutzer neben Chats auch Videokonferenzen mit bis zu zehn anderen abhalten, ohne einen externen Service wie Skype[20] zu nutzen. Diese Funktion wäre gut für den Einsatz in E-Learning nutzbar. Ein Tutor könnte so in kleiner Runde Fragestunden abhalten oder Gruppen Treffen abhalten.

2.5 Verwandte Arbeiten und Projekte

Verwandte Arbeiten Einleitung schreiben

2.5.1 What happens when Facebook is gone?

Frank McCown und Michael L. Nelson beschreiben in ihrem Bericht „What happens when Facebook is gone?“[29], wie Möglichkeiten aussehen können, die unsere Daten von sozialen Online-Netzwerken (hier im speziellen Fall von Facebook) für uns und die Nachwelt archivieren können. Zum Beispiel, wenn eine Person einen großen Teil seines persönlichen Lebens auf Facebook verbringt und plötzlich stirbt. Wie sollen seine Angehörigen an nicht öffentliche Texte, Bilder, Videos heran kommen, wenn sie in der Regel keinen Zugriff auf das Benutzerkonto haben, da der Verstorbene so etwas nicht vorhersehen konnte. Oder wenn ein Benutzer mit seinen Daten in ein anderes soziales Online-Netzwerk umziehen will, sei dies bei Facebook zum damaligen Zeitpunkt nur schwer möglich.

> *„It is also likely he was not prepared to die at such a young age, and much of his personal life, which lies in the digital “cloud“, may never be accessible to his loved ones“ [29, S. 251]*

Zum Anlegen eines solchen Archivs wurden mehrere Ansätze vorgestellt. Die einfachste Ansatz wäre die E-Mail-Benachrichtigung zu aktivieren und alle neuen Beiträge in einem E-Mail-Postfach zu sichern. So können aber nur neuen alle Beiträge erfasst werden, alte bleiben weiterhin in Facebook. Eine sehr aufwändige Möglichkeit wäre es Bildschirmfotos von den Beiträgen zu machen

[19] https://plus.google.com
[20] `http://www.skype.com/`

und diese durch ein Texterkennungsprogramm laufen zu lassen. Die dadurch erzeugten Dateien können dann in einer Datenbank gespeichert werden. Heutige Internetbrowser zusätzlich zum Anzeigen von Webseite auch der Herunterladen selbiger an. Dabei wird die HTML-Datei inklusive aller darin enthaltenen weiteren Dateien wie Bilder, Videos und CSS-Dateien gespeichert. Die so archivierte Seite hat dann im beschränkten Umfang genau das gleiche Aussehen und Verhalten wie die original Seite. Ebenfalls wäre eine Nutzung der von Facebook bereitgestellten API für Anwendungen eine Überlegung wert. 2009 war diese API noch sehr eingeschränkt. Gerade der Zugriff auf Beiträge und private Nachrichten war nicht möglich [29, S. 253, Table 1]. Für die Implementierung eines Beispiel Programms wurde ein fünfter Ansatz gewählt. Über einen sogenannten Webcrawler oder eine Erweiterung für den Browser werden relevante Seiten automatisch heruntergeladen und in einen Archiv abgelegt. Dynamische Inhalte sollen kein Problem darstellen, da Seite erst heruntergeladen wird, wenn alle Aufrufe dynamischer Funktionen abgeschlossen ist. Die archivierten Dateien können dann mittels Datamining Techniken verarbeitet und als Atom/RSS Feed[21] bereitgestellt werden.

2.5.2 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[22] 2013 ihr gestartetes Projekt „Reclaim Social“ [33] vorgestellt.

Das klingt ein wenig wie ein Werbetext ;-)

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[23]“ und der dafür vorhandenen Erweiterung „FeedWordPress“[24]. Diese Kombination ermöglichst alle Internetseiten, welche einen RSS Feed anbieten, in die Datenbank von WordPress zu spiegeln. Das Problem hierbei besteht darin, dass einige sehr beliebte Internetseiten solche RSS Feeds nicht anbieten (Facebook, Google+) oder eingestellt haben (`https://twitter.com`). Für einige solcher Seiten wurden „proxy-scripte“[33, Tech 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

[21] `http://www.rssboard.org/rss-specification`
[22] `http://re-publica.de/`
[23] `http://wordpress.org/`
[24] `http://feedwordpress.radgeek.com/`

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“[33, How Does It Work]

3 Analyse

Bevor nun ein System entwickelt werden kann, das die Synchronisation von Diskussionen auf Lernplattformen und sozialen Online-Netzwerken ermöglicht, muss erst analysiert werden, welche Aufgaben das System erfüllen muss und welche Techniken dafür geeignet sind. Alle diese Schritte sollen im folgenden Abschnitt an einen kleinen Beispiel gezeigt werden.

3.1 Ein Beispiel

Am Anfang muss irgendjemand auf einer Webseite einen Beitrag schreiben der synchronisiert werden kann. Zum Beispiel geht ein Student in das Forum seiner Veranstaltung auf der Webseite A und in den Thread zur aktuellen Übung, weil er zu ihr eine Frage hat. Er schreibt also einen neuen Beitrag in das Eingabefeld und schickt es ab. Dieser wird dann an die Webseite A geschickt, im dortigen Format (Format A) in eine Datenbank oder Ähnliches geschrieben und als neuer Beitrag im Thread angezeigt (siehe Abbildung 3.1). Da eine andere Webseite B das Format von Webseite A in der Regel nicht versteht, muss in einen nächsten Schritt der Beitrag aus der Datenbank gelesen und auf irgendeine Art konvertiert werden.

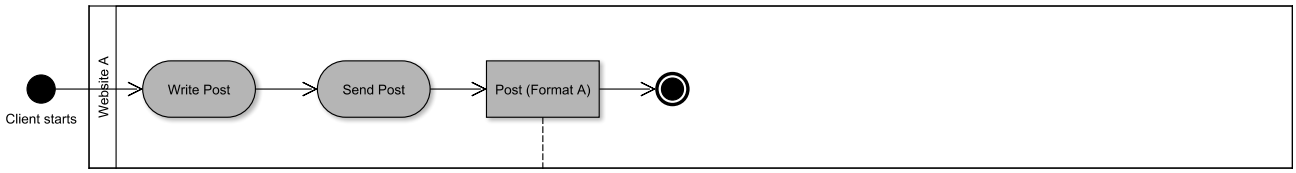


Abbildung 3.1.: Benutzer erstellt einen Beitrag im sozialen Netzwerk A.

3.1.1 Beiträge lesen und konvertieren

Abbildung 3.3 zeigt den Ablauf, wie Beiträge von der Webseite A gelesen, konvertiert und weiterverarbeitet werden. Dazu müssen zuerst die Daten über eine öffentliche API vom Server des Webseite A heruntergeladen werden. Da im Allgemeinen 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 die Webseite B geschickt werden, da sich diese in der Regel im verwendeten Datenformat unterscheiden. Diese müssen zuvor konvertiert werden.

Die einfachste Möglichkeit wäre die Daten von Webseite A, die in Format A vorliegen, in das Format B von Webseite 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 ein weiteres Webseite C unterstützt werden soll, würde ich die Anzahl an notwendigen Konvertern , wie Tabelle 3.1 zeigt, auf Sechs erhöhen.

Tabelle 3.1.: Anzahl Konverter bei drei Webseiten

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

Nimmt man an n_{ws} sei eine beliebige Anzahl sozialer Netzwerke, entspricht die Anzahl der notwendiger Konverter $n_{k1} = n_{ws} * (n_{ws} - 1)$, 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.

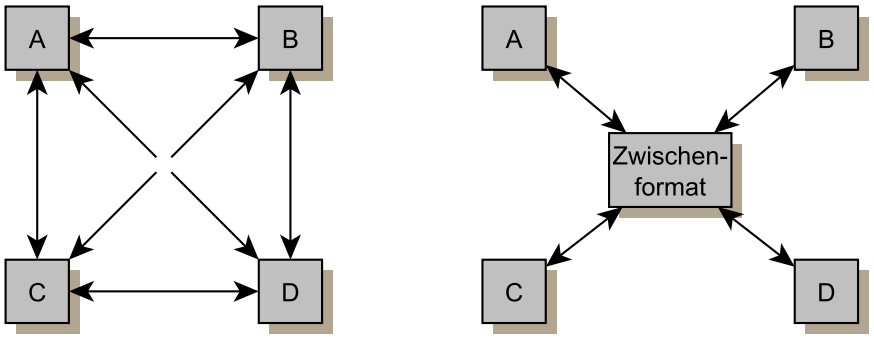


Abbildung 3.2.: Komplexität ohne und mit dem Einsatz eines Zwischenformats - Originalbild: [36]

Eine elegantere Methode für die Lösung dieses Problems, welche die Anzahl zu implementierender Konverter in Grenzen halten kann, wäre die Einführung eines Zwischenformates (auch Inter-Lingua, in Abbildung 3.3 als „IM Format“ bezeichnet) [36, S. 9]. Geht man davon aus, dass die Daten aller Webseiten nur in dieses Zwischenformat geschrieben und aus diesem gelesen werden müssen, würde sich der Aufwand auf maximal zwei Konverter je Webseite reduzieren. Für eine beliebige Anzahl Webseiten wären also $n_{k2} = n_{ws} * 2$ Konverter nötig. Nachteile hätte dieser Ansatz nur für $n_{ws} = 2$ und $n_{ws} = 3$, da in diesen Fällen mehr beziehungsweise gleich viele Konverter gegenüber der ersten Methode erforderlich wären. Erhöht man die Anzahl Webseiten jedoch nur geringfügig, sinkt die Menge an Konvertern sichtbar. Für $n_{ws} = 4$ wären es $n_{k2} = 8$ statt $n_{k1} = 12$ (siehe Abbildung 3.2) und für $n_{ws} = 5$ 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.

Bei der automatischen Sammlung von benutzergenerierten Inhalten stellt sich immer die Frage der Privatsphäre. Nicht jeder möchte, dass vielleicht sensible Informationen von ihnen weitergegeben werden. Aus diesem Grund ist die Einführung eines Mechanismus sinnvoll mit dem ein Benutzer das Lesen seiner Beiträge komplett erlaubt oder auf bestimmte Seiten, Foren oder nur für einzelne Threads beschränkt [7]. Sollte eine solche Erlaubnis nicht vorliegen, wird der gelesene Beitrag verworfen.

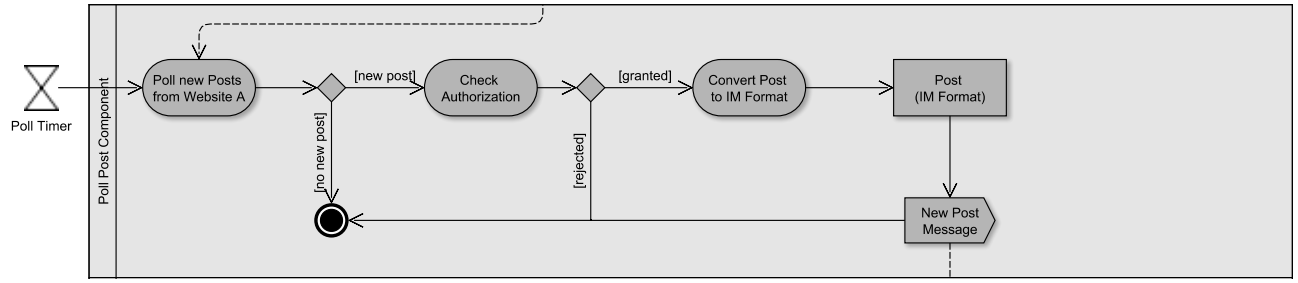


Abbildung 3.3.: Lesen des erstellten Beitrags und konvertieren in das Zwischenformat.

Da das Eintreffen neuer Beitrage nicht vorhersagbar ist, ist es angebracht beim Synchronisieren das Lesen und Schreiben zeitlich zu entkoppeln. Eine der weit verbreitetsten Techniken dazu ist das Versenden von Nachrichten über einen Nachrichtenkanal den die schreibende Komponente nach neuen Beiträgen abhört. Diesen Kanal können mehrere Gleichzeitig abhören und stellen so eine gute Flexibilität sicher.

3.1.2 Beiträge in eine andere Webseite schreiben

Empfängt nun eine Komponente, die für das Schreiben zuständig ist, eine Nachricht von einen neuen Beitrags der Webseite A ist der Ablauf ähnlich wie beim Lesen nur in umgekehrter Reihenfolge (Siehe Abbildung 3.4 links, oberer Ablauf). Der neue Beitrag wird vor dem Schreiben aus dem Zwischenformat in das Format B konvertiert. Da bei einer Synchronisation vom Vorteil wäre, wenn der synchronisierte Beitrag so aussehen würde, als hätte ihn der original Autor geschrieben. Hierzu muss das System Zugriff auf die Benutzerkonto haben und diese in einer Datenbank verwalten. Dort kann dann nach einen passenden Benutzerkonto des Autors gesucht und dieses dann zum Schreiben verwendet werden. Steht ein solches Benutzerkonto nicht zur Verfügung, ist das Ausweichen auf ein vorgegebenes Benutzerkonto hilfreich. Manche Benutzer sind vielleicht damit einverstanden, dass ihre Beiträge weitergegeben werden, aber nicht dass andere Programme automatisch in ihren Namen Beiträge schreiben. Deswegen ist wichtig davor noch einmal zu prüfen, ob eine entsprechende Erlaubnis gegen wurde. Sind alle Voraussetzungen gegeben, kann der Beitrag über das ausgewählte Benutzerkonto auf die Webseite B geschrieben werden.

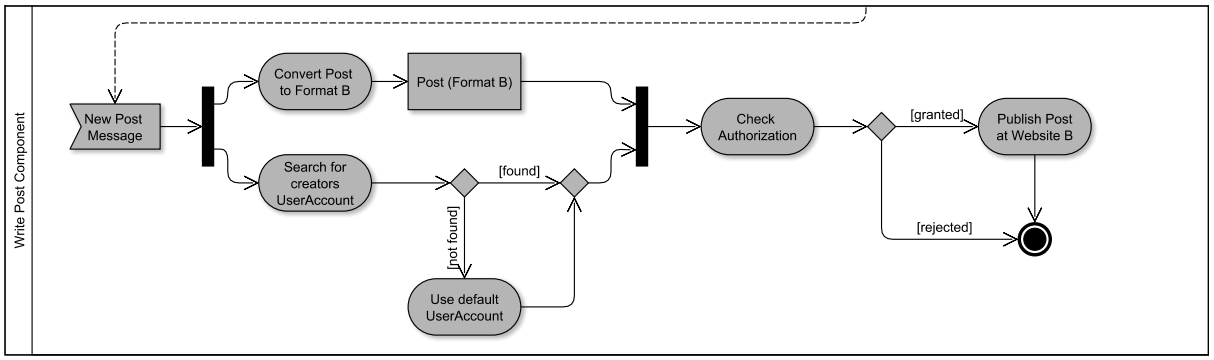


Abbildung 3.4.: Konvertierten des Beitrags in das Format B und schreiben n das soziale Netzwerk B

3.2 Identifizierung der Komponenten

Anhand dieses kurzen Ablaufbeispiels kann man einige Komponenten ablesen, aus denen das neue System auf jeden Fall bestehen beziehungsweise enthalten muss und welche ergänzend dazu Wünschenswert wären:

- Eine Komponente muss Daten von einen Webseite über deren öffentlicher API in das System einlesen und diese in ein geeignetes Zwischenformat konvertieren können.
- Es muss ein passendes Nachrichtensystem zum Entkoppeln von Lesen und Schreiben gewählt und/oder erstellt werden.
- 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 dem Konto eines Benutzer zu einer bestimmten Webseite suchen zu können.
- Um die Privatsphäre der Benutzer zu wahren, wäre eine Mechanismus zum festlegen von Zugriffsrechen sinnvoll.

3.3 Wahl vorhandener Techniken

vielleicht doch eher an den Anfang von Ëigener Ansatz"!?

3.3.1 Welches Zwischenformat?

- SIOC + FOAF als geeignetes Zwischenformat
- warum Ontologie und keine neues XML Schema!?

3.3.2 Welches Nachrichtensystem?

JMS

- zum reinen verschicken ausreichend
- fast alles muss selber gemacht werden.

Apache Camel

- erweitern der "Pipelineß.B. mit Filtern einfach möglich

- gute kombination -> durable subscriber

4 Eigener Ansatz: Social Online Community Connectors (SOCC)

Aufbauend auf den in Kapitel 3 identifizierten Komponenten und Wahl der für ein System zur Synchronisation von Beiträgen passenden Techniken, soll nun der als *Social Online Community Connectors* (SOCC) benannter Ansatz vorgestellt werden. SOCC setzt für seine Aufgabe auf Techniken des in den Grundlagenkapitel beschriebenen Semantic Webs. Der Einsatz von offenen Datenformaten wie RDF und den darauf aufbauenden FOAF und SIOC nicht nur für den hier beschrieben Zweck sondern auch für andere Projekte verwendet werden kann.

Der Aufbau von SOCC zeig die Abbildung 4.1. Ein *Connector* des SOCC dient als Verbindungselement zwischen zwei oder mehr Webseiten, die unter dem Begriff *Social Online Community* (SOC)zusammengefasst werden. Also einer Gemeinschaft die im Web auf soziale Weise Wissen austauscht. SOCC stützt sich dabei dabei auf die Prinzipien für die Verbindung von Daten im Web von Tim Berners-Lee [4]:

„Use URIs as names for things" Seiten, Foren, Threads und Beiträge sollten immer über eine URI benannt werden, so das andere Anwendungen diese verwenden können.

„Use HTTP URIs so that people can look up those names" Die verwendeten URIs sollten dereferenzierbar sein, um auf die dahinter liegenden Daten zugreifen zu können.

„When someone looks up a URI, provide useful information, using the standards (RDF*, SPARQL)" Die Daten sollten in einen Standard für das Semantic Web formatiert werden. Wie schon wähnt setzt SOCC dazu auf RDF und darauf aufbauenden Ontologien wie FOAF und SIOC. Dadurch können zum Beispiel andere Anwendungen mit Anfragen in SPARQL[1] nach Beiträgen suchen.

„Include links to other URIs. so that they can discover more things." Nicht nur die Struktur eine Diskussion kann über Links verfolgt werden, auch das Gewinnen zusätzlicher Informationen ist so möglich. Beiträge können auf Lernmaterialien wie Folien oder Videos verweisen oder über das FOAF-Profil eines Autor können weitere Beiträge von ihm gefunden werden.

URIs sind also das wichtigste Element mit denen SOCC arbeitet. Soll zum Beispiel eine Diskussion synchronisiert werden, muss einem Connector die URI übergeben werden, hinter der sich die Daten befinden. Ein Connector ist nur für eine einzige SOC zuständig. Aber eine SOC kann über mehrere Connectoren angesprochen werden, wodurch eine Arbeitsteilung möglich ist.

Intern besteht ein Connector aus drei Teilkomponenten die zum Einen für das Lesen (*PostReader*) und Schreiben (*PostWriter*) von Beiträgen verwendet werden, zum Andren aus einen *StructureReader* der für das auslesen der Struktur der einzelnen Diskussionen verantwortlich ist. Eine genaue Beschreibung dieser Komponenten folgt in Abschnitt 4.3.

[1] `http://www.w3.org/TR/sparql11-overview`

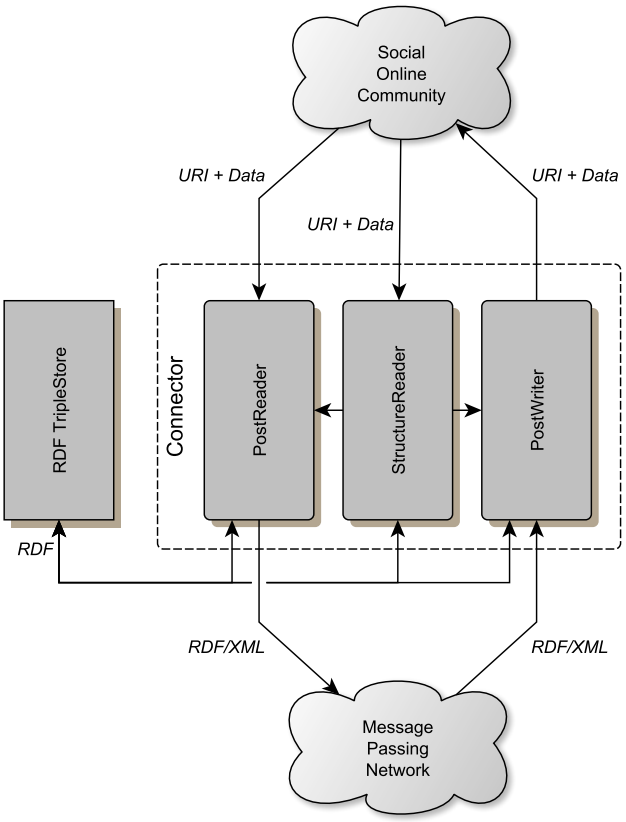


Abbildung 4.1.: Übersicht der Komponenten der SOCC

Jeder Connector hat Zugriff auf eine als *Triplestore* bezeichnete Datenbank in der RDF-Triple gespeichert und unter anderen mit SPARQL abgefragt werden. Der Connector benutzt diesen Triplesore als Speicher in dem seine Konfigurationsdaten lagern, aber auch zusätzliche Daten die er von Außerhalb, wie eine Liste von Benutzerkonten, benötigt. Er wird aber auch benutzt um Daten zu speichern die während des Betriebes anfallen, da sie so irgendwann wieder verwendet werden können ohne sie erneut zusammenzusuchen. Zum Beispiel die Daten über die Struktur der verwendeten SOC.

Die Beiträge werden dann zwischen den einzelnen Connectoren über ein Nachrichtennetzwerk auf der Basis von Apache Camel ausgetauscht. Die Beschreibung dieser als *SOCC-Camel* bezeichneten Komponnente geschieht in Kapitel 4.4.

4.1 Datenformat

Auch begründen warum die Verwendung von Ontologien und nicht bspw. die Entwicklung eines XML Schemas (alternativ bei der Analyse Kap. 3.2)

4.2 Konfiguration

Dass ein Connector funktionieren kann, muss er von außen mit Informationen zugeführt bekommen welche er zum Betrieb braucht. Die sind zum Beispiel Informationen zu Benutzerkonten oder Parameter für die verwendete API. Da einige dieser Informationen werden nicht nur von einen Connector benutzt werden, ist es sinnvoll diese zusammen an einen Ort zu speichern und wiederverwenden zu können. Die wichtigsten Informationen für die Konfiguration der Connectoren stellen die Benutzerkonten dar. Sie enthalten unter anderem die Informationen um Zugriff auf die einzelnen APIs zu erhalten. Da die Benutzerkonten wie im Abschnitt 4.1 beschrieben im FOAF Format in einen Triplestore gespeichert werden, stellt es sich als Vorteil heraus die übrigen Informationen ebenfalls dort zu speichern und mit den schon vorhandenen zu verbinden.

Aus diesem Grund wurde für Konfiguration eines Connectors die *Connector Config Ontology* entwickelt. Diese Ontologie ist sehr einfach gehalten und baut auf schon vorhandenen Ontologien auf. Zusätzlich musste die SIOC Ontologie so erweitert werden, dass die Integration von Autorisierungs- und Authentifizierungsinformationen möglich war.

4.2.1 SOCC Connector Config Ontologie

Abbildung 4.2 zeigt die entwickelte *SOCC Connector Config* Ontologie. Sie besteht aus einer einzigen Klasse `ConnectorConfig` und fünf Eigenschaften für diese.

Einleitung umschreiben, klingt scheiße

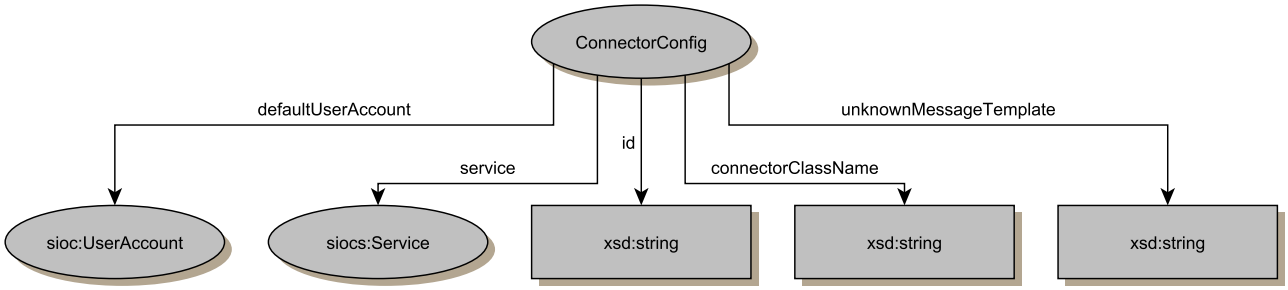


Abbildung 4.2.: Schema der SOCC Connector Config Ontology

Jeder Connector erhält einen eindeutigen `id` zugewiesen, um jeden Connectoren später eindeutig identifizieren zu können. Die Eigenschaft `connectorClassName` beschreibt den vollständigen Klassennamen des beschriebenen Connector. Diese wird für das laden der richtigen Implementierung benötigt. Manchmal kann es passieren, dass keine passendes Benutzerkonto zum Weiterleiten eins Beitrags gefunden werden kann. Dadurch es es wünschenswert solche Beiträte dahingegen zu verändern, dass eine Verweis auf den original Autor und vielleicht wo der Beitrag gemacht wurde vorhanden ist. Durch die Eigenschaft `unknownMessageTemplate` kann eine Vorlage für das Aussehen des des Verweises definiert werden.Innerhalb dieser Vorlage stehen einige Variablen in der Form `{varName}` zur Verfügung. Alle zur Zeit vorhandenen Variablennamen und deren Ersetzung sind in Tabelle 4.1 zu sehen.

Tabelle 4.1.: Variablennamen und Ersetzung innerhalb von `unknownMessageTemplate`

varName	Ersetzt durch
`message`	Original Beitrag
`sourceUri`	URI des original Beitrags
`connectorId`	ID des aktuellen Connectors
`serviceName`	Name des vom Connector verwendeten Service
`creationDate`	Erstelldatum des Beitrags (falls bekannt)
`authorName`	Name des Autors (falls bekannt)

Für die Nutzung einiger APIs müssen bestimmte Parameter angegeben werden. Dies könnte zum Beispiel die genau Adresse des Dienstes sein. Hierzu wird auf die schon Bestehende SIOC Services Ontologie zurückgegriffen. Diese stellt eine Klasse *Service* zur Verfügung und mittels der Eigenschaft `service` kann ein solcher Service einem Connector zugewiesen werden. Der genaue Aufbau eines solchen Services wird im Abschnitt 4.2.2 dargestellt. Die letzte Information für die Konfiguration eins Connectors ist eine vordefinierter Benutzer (im Folgenden Defaultuser genannt) und wird mit der Eigenschaft `defaultUserAccount` festgelegt. Dieser Defaultuser erfüllt im Großen und Ganzen zwei Aufgaben. Als Erstes wird er für lesende Zugriffe der API auf den verwendeten Dienst genutzt. Hierzu ist ein einzelnes Benutzerkonto vollkommen ausreichend, da nur die gelesenen Daten wichtig sind und nicht von welchen Konto sie kommen.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 Defaultuser genutzt und der Beitrag mit einem Vermerk zum original Autor über diesen geschrieben.

4.2.2 Services

Wie eben schon beschrieben, existiert für SIOC ein Modul zur einfachen Modellierung von Diensten auf semantischer Ebene: Das SIOC Services Module (Präfix *siocs:*). Kernstück dieses Moduls ist die Klasse Service, wie auf Abbildung 4.3 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. 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, SOAP, ...) und `service_definition` mit dem auf eine weiterführende Definition verwiesen werden kann.

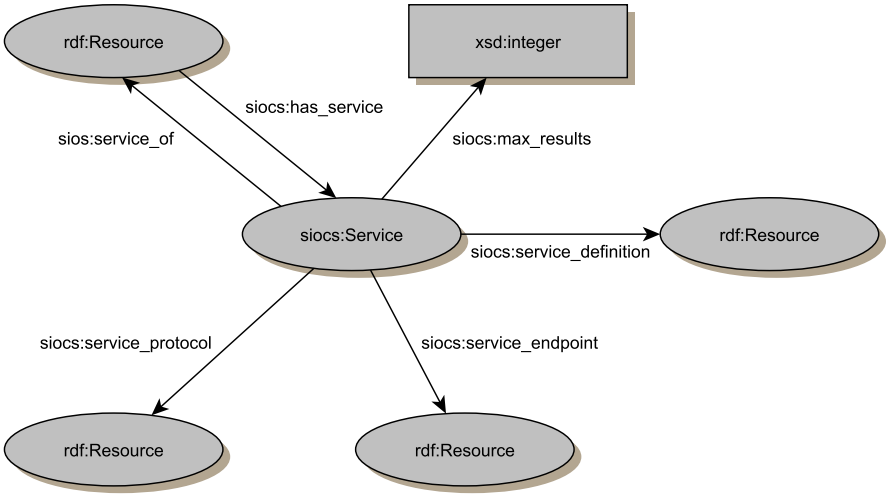


Abbildung 4.3.: SIOC Services Module

4.2.3 Benutzerdaten

Soll ein Beitrag eines Benutzers von Google+ nach Facebook synchronisiert werden und es so aussehen, als hat er diesen Beitrag selbst auf Facebook geschrieben, sind gute Kenntnisse über alle Benutzerkonten dieser einen Person notwendig. Als erstes muss die Existenz dieser Person dem System bekannt sein. Hierzu kann diese durch die Klasse `Person` aus der FOAF Ontologie dargestellt werden. Für ein einzelnes Benutzerkonto wurde in SIOC die Klasse `UserAccount` definiert. Da `UserAccount` eine Unterklasse von `OnlineAccount` aus FOAF ist, kann diese über die Eigenschaft `foaf:account` beziehungsweise `sioc:account_of` mit einer Person verbunden werden. Da es wichtig ist zu wissen zu welcher Webseiten ein Benutzerkonto gehört, wird der `UserAccount` mit einem Objekt der Klasse `Service` über die Eigenschaft `siocs:has_service` `/siocs:service_of` zusammengebracht. Diese Verbindung ist für manche APIs besonders bedeutend, da in dem Serviceobjekt relevante Daten für den Zugriff darauf enthalten sind. Nun kann es vorkommen, dass eine Person private und geschäftliche Benutzerkonten besitzt. Um nicht private Beiträge auf Webseite A mit dem geschäftlichen Benutzerkonto auf Webseite B zu schreiben, muss ein Mapping zwischen den verschiedenen Benutzerkonten festgelegt werden. Dieses Mapping kann über ein in der „SOCC Connector Config Ontologie“ definierte Eigenschaft `mapped_to` realisiert werden. Diese Eigenschaft ist bijektiv, also jedes Benutzerkonto maximal auf ein anderes Konto gemappt werden darf. Ebenso gilt, falls Benutzerkonto A mit Benutzerkonto B verbunden ist, ebenfalls B nach A gemappt wird. Abbildung 4.4 zeigt den zusammenhang zwischen der Klasse `Person`, `UserAccount` und `Service` sowie der Eigenschaft `mapped_to` noch einmal graphisch an einem Beispiel.

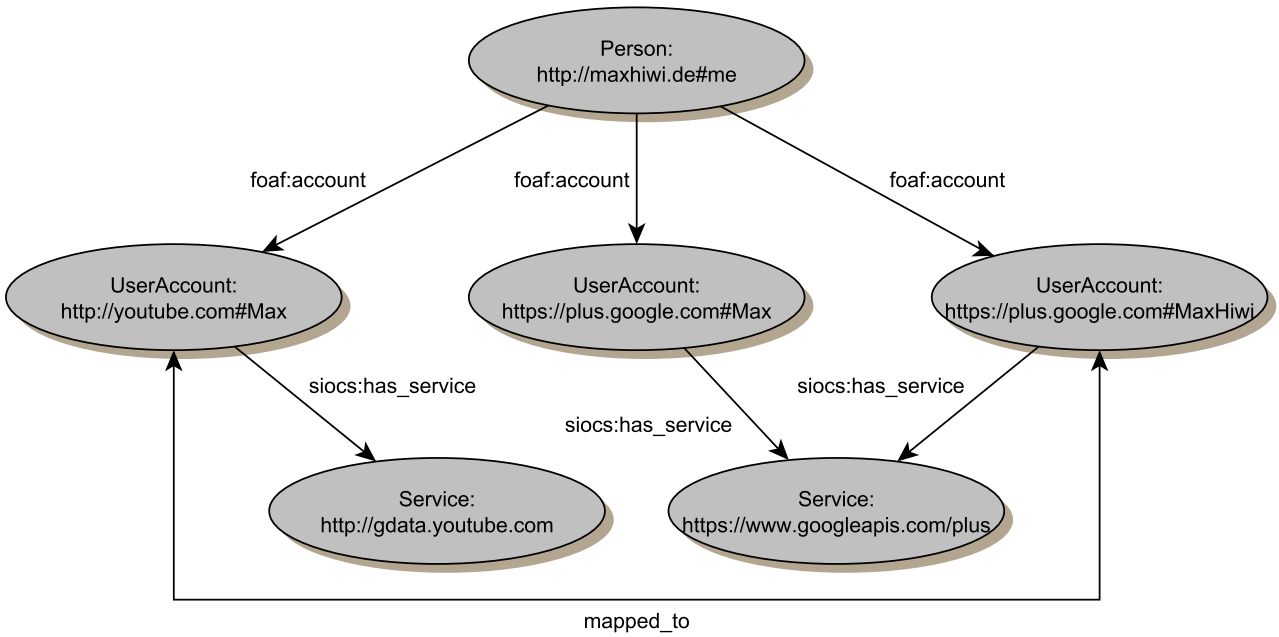


Abbildung 4.4.: Zusammenhang von Person, UserAccount und Service. Die inversen Eigenschaften `sioc:account_of` und `siocs:service_of` wurden zu einer besseren Übersicht weggelassen

4.2.4 Authentifizierung

Die Ontologien FOAF und SIOC sind hervorragend für die Abbildung von sozialen Netzwerken und Diskussionen, jedoch ist es mit ihnen nicht möglich Daten zur Authentifizierung (Feststellen ob jemand der ist, den er vorgibt zu sein). Wegen dem Schutz der Privatsphäre ist dies verständlich, jedoch um stellvertretend für einen Benutzer Beiträge zu schreiben, ist es wichtig Zugriff auf diese Daten zuhaben und sie einem Benutzerkonto zuordnen zu können. Zuerst muss dazu aber festgestellt werden, welche verschiedenen Mechanismen es zum Anmelden ein solches Konto existieren.

Username/Passwort ist wohl eine der ersten und häufigsten Mechanismen, um den Zugriff sensibler Daten vor Dritten zu schützen. Das in Abschnitt 5.2.1 beschriebene LMS Moodle, setzt zum Beispiel den Username und Password eines angemeldeten Benutzers zu Authentifizierung ein.

OAuth [2] 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[3]. 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* [20][S. 8]. Sie weisen das Programm als autorisiert für die Benutzung der

[2] `http://oauth.net/`
[3] `http://en.wikipedia.org/wiki/OAuth#List_of_OAuth_service_providers`

Schnittstelle aus. Soll nun beim Aufrufer einer von OAuth geschützten Funktion belegt werden ist ein sogenannter Accesstoken[20][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 5.2.4 beschriebene Google Youtube API hierzu ein gutes Beispiel.

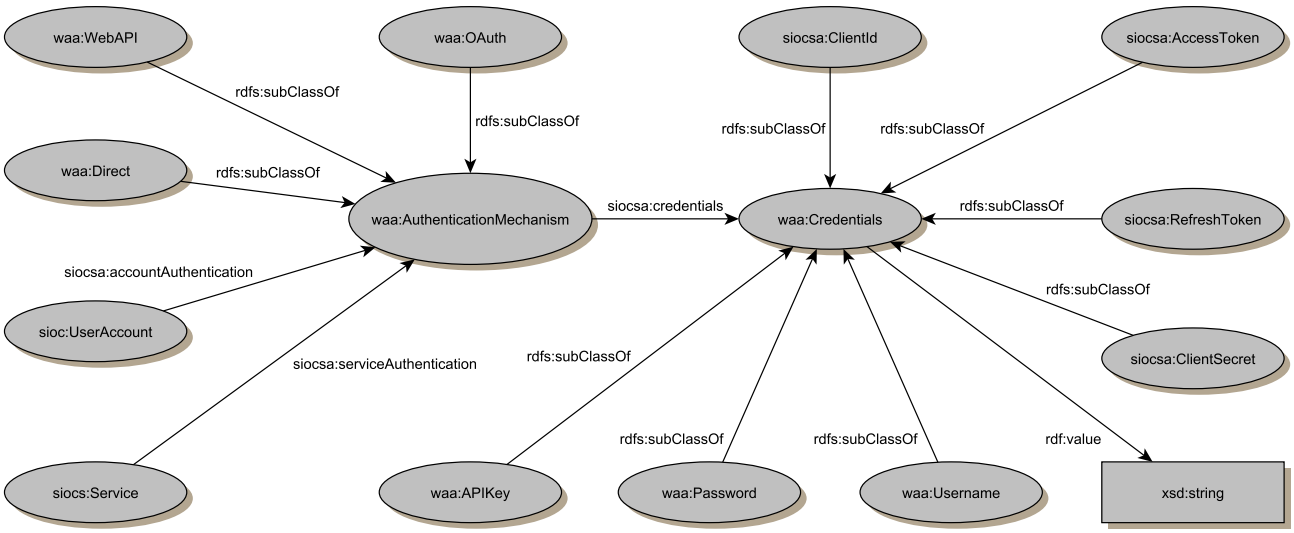


Abbildung 4.5.: SIOC Services Authentication Ontology

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 dem 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 zum Beispiel die *Web Access Control List* [24] mit Zugriffssteuerungsliste. Eine Ausnahme stellt die *Authentication Ontology*[4] des *OmniVoke*[5] Frameworks dar. Die Art der Authentifizierung wird darin durch die Klasse `AuthenticationMechanism` modelliert. Unterklassen für die wichtigsten Mechanismen wie `OAuth`, `WebAPIs` und Userna-

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

me/Password (dort `Direct` genannt) sind vorhanden. Jedem AuthenticationMechanism Objekt können dann sogenannte `Credentials` (engl. für Anmeldedaten) angehängt werden.

Das einzige Manko an dieser Ontologie war das Fehlen von Credentials für OAuth in der Version 2.0. Im einzelnen waren dies Klassen für clien_id, client_secret sowie für Access- und Refreshtoken. Um auch diese OAuth Version unterstützen zu können, wurden hierfür die Klassen ClientId, ClientSecret, AccessToken und RefreshToken als Unterklassen von Credentials abgeleitet. Als Letztes musste noch eine Verbindung zwischen Authentication Ontology und SIOC hergestellt werden. Zum Einen war eine Erweiterung der Klasse UserAccount notwendig, so dass die Anmeldedaten der Benutzer zur Verfügung standen. Zum Anderen werden Daten wie ein API Schlüssen von einen Service benötigt, die von denen der Benutzer unabhängig sind. Für die Klasse UserAccount wurde die Eigenschaft `accountAuthentication` geschaffen. Diese erwartet als Subjekt einen UserAccount und als Objekt ein AuthenticationMechanism, welcher dann die Credentials enthält. Für die Klasse Service existiert das Äquivalent `serviceAuthentication`.

Diese Erweiterungen (Präfix *siocsa:*) und die übernommenen Teile der Authentication Ontology wurden danach im *SIOC Services Authentication Module* zusammengefasst. Graphisch ist sie in Abbildung 4.5 und im Anhang A.2 als OWL Schema zu sehen.

4.2.5 Autorisierung

Da für viele Menschen im Internet ihre Privatsphäre und nicht wollen, dass ohne ihr Wissen Informationen von ihnen weitergegeben werden, ist es wichtig von den Benutzern die Erlaubnis zu holen Beiträge weiter zu leiten. Ein verbreitetes Mittel für eine solche Zugriffssteuerung sind Access Control Lists (ACL) (engl. für Zugriffsteuerungsliste). Die anderen nur einen begrenzten Zugriff auf eine feste Ressource gewähren. Für den Einsatz in dieser Arbeit wurde die Basic Access Control Ontologie [24, 1] ausgewählt (Im Folgenden nur als ACL bezeichnet). Der Hauptgrund lag darin, dass sie auf FOAF aufbaut und sich so einfach integrieren ließ.

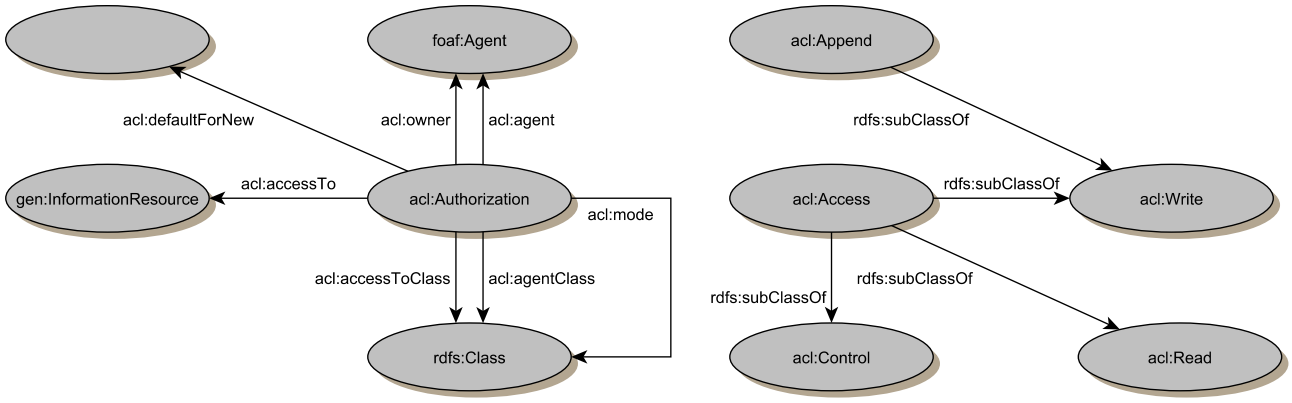


Abbildung 4.6.: Basic Access Control Ontologie

Die ACL stellt die Klasse `acl:Access` für die Abbildung eines Zugriffrechts auf eine Ressource zur Verfügung. Von dieser Klasse werden einzelne Rechte wie `acl:Read` für das Lesen und `acl:Write` für das Schreiben abgeleitet. Nebenbei existieren noch Ableitungen `acl:Control`

zum Ausführen und `acl:Append` zum Anfügen von Inhalten. Diese sind für diese Arbeit aber nicht von Bedeutung.

Die eigentliche Regel für die ACL besteht aus der Klasse `acl:Authorization`. Der Besitzer dieser Regel wird durch die Eigenschaft `alc:owner` festgelegt. Der Besitzer ist im Fall von SOCC ein Objekt der Klasse Person, wie in Abschnitt 4.2.3 „Benutzerdaten“ beschrieben, da diese eine Unterklasse von der ACL geforderten `foaf:Agent` ist. Mittels `acl:agent` kann signalisiert werden, dass diese Regel nur für eine bestimmte Person/Agenten gilt. Das selbe gilt für `acl:agentClass`, wobei hier eine bestimmte Klasse gemeint ist. Soll zum Beispiel ein öffentlicher Zugriff definiert werden, wird `foaf:Agent` eingesetzt [1, „Public Access“]. Innerhalb von SOCC werden nur Regel angewendet, die einen solchen öffentlichen Zugriff erlauben. Die eigentlichen Rechte werden über die Eigenschaft `acl:mode` festgelegt. Erlaubt ist die Angabe jeder beliebigen Klasse, SOCC testet aber nur auf die Klassen `acl:Read` und `acl:Write` zum Lesen und Schreiben von Beiträten. Auf welche Ressource sich ein Regel letztendlich bezieht, wird über die Eigenschaft `acl:accessTo` geregelt. Die Angabe von „http://www.facebook.com“ würde sich für SOCC zum Beispiel auf alle Beiträge des Besitzers auf Facebook beziehen, „https://canvas.instructure.com/courses/798152“ dahingegen nur auf alle Beiträge innerhalb eines Canvas Kurses. Für einen Zugriff auf alle Beiträge prüft SOCC ob die Eigenschaft `acl:accessToClass` auf die Klasse `sioc:Post` verweist. So müsste nicht jede einzelne Seite angegeben werden.

Das Listing 4.1 zeigt ein Beispiel wie ein ACL Regel aussehen könnte. Der Besitzer dieser Regel wird durch die URI `http:/example.org#john` beschrieben (Zeile 6). Diese Person erlaubt nun öffentlichen Zugriff (Zeile 7) all all seine Beiträge (Zeile 8). Dieser Zugriff kann sowohl lesend als auch schreibend erfolgen (Zeile 9).

Listing 4.1: ACL Beispiel

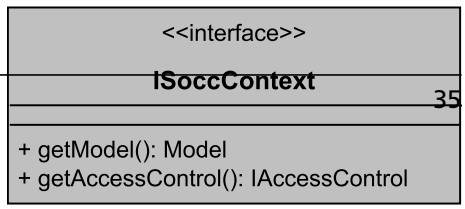
```
1 @prefix sioc: <http://rdfs.org/sioc/ns#> .
2 @prefix foaf: <http://xmlns.com/foaf/0.1/>
3 @prefix acl: <http://www.w3.org/ns/auth/acl#> .
4
5 [] a acl:Authorization;
6     acl:owner <http:/example.org#john>;
7     acl:agentClass foaf:Agent;
8     acl:accessToClass sioc:Post;
9     acl:mode acl:Read, acl:Write .
```

4.3 Design eines Connectors

Einleitung für design schreiben

4.3.1 SOCC Context

Der Kontext eines Connectors beschreibt die Umgebung innerhalb dem er seine Arbeit verrichtet. Im aktuellen Status



erhält der Connector zum einen Zugriff auf ein außenstehendes Model (TripleStore), das wichtige Daten für den Betrieb und enthält oder abgelegt werden können. Eine Referenz auf dieses Model erhält der Connector über den Aufruf der Funktion `getModel()`. Durch die Methode `getAccessControl()` kann der Connector über die im nächsten Abschnitt beschriebene AccessControl Schnittstelle auf die Information für die Zugriffssteuerung für das Lesen und Schreiben von Beiträgen.

4.3.2 AccessControl

Die AccessControl Schnittstelle ist sehr einfach gehalten und dient für den Zugriff auf die in Abschnitt 4.2.4 „Autorisierung“ beschriebene ACL. Die Methode `checkAccessTo(...)` prüft, ob der Zugriff auf eine Ressource mit allen übergebenen Zugriffsmodi erlaubt ist. Die andere Methode `checkAccessToClass` ist zur Überprüfung, die die Rechte für den Zugriff auf eine komplette Klasse von Ressourcen.

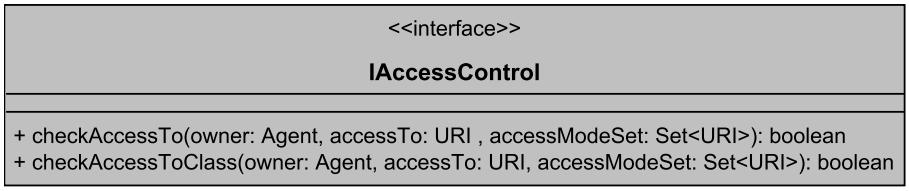


Abbildung 4.9.: AccessControl

4.3.3 ClientManager

Der Zugriff auf eine API innerhalb eines Programms erfolgt in der Regel über eine sogenanntes Clientobjekt (kurz Client). Dieser Client erlaubt es mit den Anmeldedaten oder den Accesstoken für ein Benutzerkonto auf die Funktionen der API über verschiedene Methoden zu zugreifen. Da ein Client immer nur mit einem Benutzerkonto verknüpft ist und von diesen eine große Anzahl verwaltet werden müssen, enthält jeder Connector einen ClientManager. Der interne Aufbau eines Client ist dabei stark von der verwendeten API abhängig und arbeitet mit dem Connector zusammen für den er geschrieben wurde. Für alle benutzerunabhängigen Daten erhält der ClientManager ein wie Abschnitt 4.2.2 beschriebenes Serviceobjekt. Ein neuer Client kann dann durch den Aufruf der Methode `createClient(...)` erstellt werden. Als Parameter wird der Methode ein Benutzerkonto in Form eines SIOC UserAccounts übergeben. Sind alle erforderlichen Autorisierung- und Authentifizierungsinformation

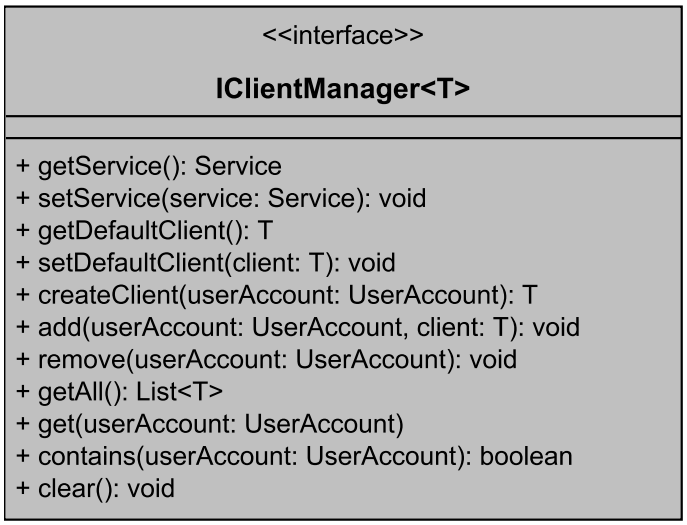


Abbildung 4.10.: ClientManager

aus Abschnitt 4.2.4 vorhanden, wird ein neuer Client erzeugt und zurück gegeben. Dieser Client wird aber dadurch nicht automatisch vom ClientManager verwaltet. Hierzu muss der im vorherigen erzeugte Client durch den Aufruf von `add(userAccount: UserAccount, client: T )` dauerhaft mit den angegeben UserAccount verknüpft und intern gespeichert. Wichtig ist hierbei, dass die Eigenschaften `accountName` und `accountServiceHomepage` des UserAccount Objekts gesetzt sind. Aus diesen wird ein eindeutiger Schlüssel generiert der zur Zuordnung von UserAccount und Client innerhalb des ClientManagers dient. Des weiteren stehen noch Methoden `remove(userAccount: UserAccount)` zum Entfernen und `get(userAccount: UserAccount)` Holen von Clients, sowie `contains(userAccount: UserAccount)` für Tests ob ein Client zu einem UserAccount existiert. Sollen zum Beispiel am Ende der Laufzeit des Programms alle erzeugten Clients auf einmal abgemeldet und gelöscht werden, kann dies über die Methode `clear()` erfolgen. Der ClientManager verwaltet ebenfalls den Client für den in Abschnitt 4.2.1 angesprochenen Defaultuser. Dieser Defaultclient genannte Client kann über die Methode `setDefaultClient(client: T)` gesetzt und durch `getDefaultClient()` jederzeit wieder abgerufen werden.

4.3.4 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 2.2.2 vorgestellt, durch die SIOC Klassen *Site* und *Container* (und Unterklassen davon) beschrieben. Um auf diese Struktur zugreifen zu können, enthält die StructureReader Schnittstelle mehrere Methoden (Siehe Abbildung 4.11).

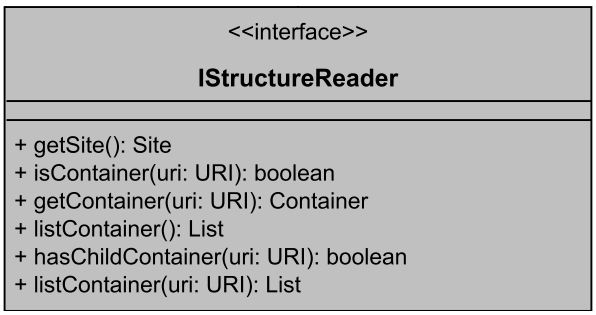


Abbildung 4.11.: StructurReader

`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.

`isContainer(uri: URI)` wird zu Überprüfung verwendet, ob sich hinter einer URI ein potenzieller Container befindet.

`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: 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.

4.3.5 PostReader

Der `PostReader` dient als Schnittstelle für das Lesen geschriebener Beiträge innerhalb eines Containers oder der Kommentare auf einen anderen Beitrag. Er stellt nach außen hin Funktionen bereit mit denen entweder ein einzelner Beitrag oder alle Beiträge die bestimmte Kriterien erfüllen gelesen werden können. Bevor ein Beitrag zurück gegeben wird, müssen die Methoden prüfen, ob der Autor dieses Beitrag das Lesen dafür erlaubt hat. Falls nicht, wird der Beitrag aus der Ergebnisliste gelöscht oder ein Fehler ausgegeben. Die Funktionsweise der einzelnen Methoden ist wie folgt:

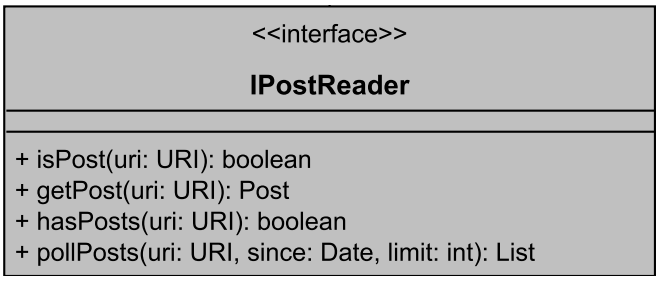


Abbildung 4.12.: PostReader

`isPost(uri: URI)` kann zur Überprüfung eingesetzt werden, ob sich hinter einer URI ein Beitrag befindet.

`getPost(uri: URI)` ist dazu gedacht einen einzelnen Beitrag anhand seiner URI zu lesen. Sie liefert dann den Beitrag SIOC Post Objekt zurück beziehungsweise einen Fehler, falls der Beitrag nicht mit diesem Connector gelesen werden kann.

`hasPost(uri: URI)` funktioniert ähnlich wie isPost, überprüft aber ob sich hinter der angegeben URI noch weitere Beiträge befinden.

`pollPosts(uri: URI, since: Date, limit: int)` ist eine Methode mit der alle Beiträge hinter eine URI liest und anhand der übergebenen Kriterien filtert. Insgesamt erhält diese Methode drei Parameter. 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. Zum Beispiel der Zeitpunkt als diese Methode das letzte mal aufgerufen wurde, um alle Beiträge die danach folgten zu lesen. Der letzte Parameter gibt eine obere Schranke an, wie viele Beiträge maximal pro Aufruf dieser Methode gelesen werden dürfen.

4.3.6 PostWriter

In Abbildung 4.13 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 Defaultclient zur Verfügung. Mit diesem Client, ob mit der des Autors oder dem Defaultclient, wird im letzten Schritt der Beitrag im von der API verwendeten Format in die SOC geschrieben.

unknownMessageTemplate und Watermark erwähnen

4.4 SOCC-Camel

SOCC-Camel ist eine Modul der SOCC für die Einbindung von Connectoren in das Apache Camel Framework. Durch dieses ist es sehr einfach möglich die gelesenen Beiträte von einer SOC in eine andere zu schreiben. Abbildung 4.14 zeigt SOCC-Camel als EIP-Diagramm.

Hauptklasse dieses Modul ist die Klasse `SoccComponent` und ist für die Verwaltung von Connectoren und Erstellen der Endpoints für Apache Camel zuständig. Von dieser Klasse SoccComponent muss zuerst ein Objekt erzeugt und ein Objekt einer Klasse welche die Schnittstelle `ISoccContext` implementiert. Das das damit übergebende Model mit dem Triplestore sollte alle Daten für die Erstellung der später genutzten Connectoren und deren Betrieb enthalten. Also alle Daten die in Anschnitt 4.2 beschrieben sind. Das nun vorhandene SoccComponent Objekt wird dann unter einen frei wählbaren Namen in Apache Camel registriert. Nun kann wie in Abschnitt 2.3.2 mit dieser Komponente eine Route erstellt werden. Die Konfigurationsuris habe dazu den folgenden Aufbau:

Listing 4.2: SOCC-Camel Konfigurations URI

```
1 socc://{connectorId}?uri={targetUri}[&{options}...]
```

Der Anfang der URI mit „socc://“ sagt Apache Camel, dass es für diesen Endpoint die Component benutzten soll, die unter dem Namen „socc“. Hier wäre dies eine der Klasse SoccComponent. Für den Platzhalte „{connectorId}“ muss eine gültige ID eines Connectors sein, dessen Konfigurationsdaten im TripleStore. befinden. Über den URI Parameter „uri“ wird dann die Quell- beziehungsweise Zieluri für den Connector angegeben. Im Falle, dass es sich um eine URI für den Startpunkt einer URI handelt, können am Ende noch weitere Parameter angehängt werden.

4.4.1 SoccPostPollingConsumer

Wird ein Endpoint mit der Absicht zum Lesen von Beiträgen erstellt, erzeugt die Klasse SoccComponent eine Objekt der Klasse `SoccPostPollingConsumer` und übergibt im die Parameter aus der verwendeten URI. Als zusätzliche Parameter für die URI können `delay` und `limit angegeben werden`. Da SoccPostPollingConsumer sich von der Klasse `ScheduledPollConsumer` ableitet, ist

es über den Parameter `delay` möglich in periodischen Abständen Aufgaben abzuarbeiten. In diesen Fall das Lesen von neuen Beiträgen. Die Angabe erfolgt dabei in Millisekunden. Der Parameter `limit` entspricht dabei den gleichnamigen Argument der Methode `pollPosts` des PostReaders aus Abschnitt 4.3.5. Das dort noch fehlende Argument `since`, für das Datum ab wann ein Beitrag als neu gilt, holt sich der SoccPostPollingConsumer aus den im TripleStore gespeicherten Daten. Zum Beispiel durch das Datum des zuletzt gelesenen Beitrags aus einen Thread oder das des letzten Kommentars. Alle neue gelesenen Beiträge werden am Ende in das RDF/XML serialisiert und als Nachricht an Apache Camel übergeben, welches es dann anhand der festgelegte Routen weiterleitet. Dass andere Komponenten diese Nachricht wieder korrekt deserialisieren können, muss der Nachricht noch ein Header „Content-Type“ mit dem MIME-Type[6] vom RDF/XML „application/rdf+xml“ mitgegeben werden

4.4.2 SoccPostProducer

Der `SoccPostProducer` ist das Gegenstück zum SoccPostPollingConsumer er ist der Endpoint der zum Schreiben von Beiträgen zurück eine ein SOC von der SoccComponent erzeugt wird. Intern verwendet er die PostWriter Komponente des betreffenden Connetors und leitet den Inhalt der Nachricht an diese weiter. Außer der Angabe über die Zieluri erhält der SoccPostProducer keine weiteren Parameter über die Konfigurationsuri.

[6] Multipurpose Internet Mail Extension - Types: `http://tools.ietf.org/html/rfc2046`

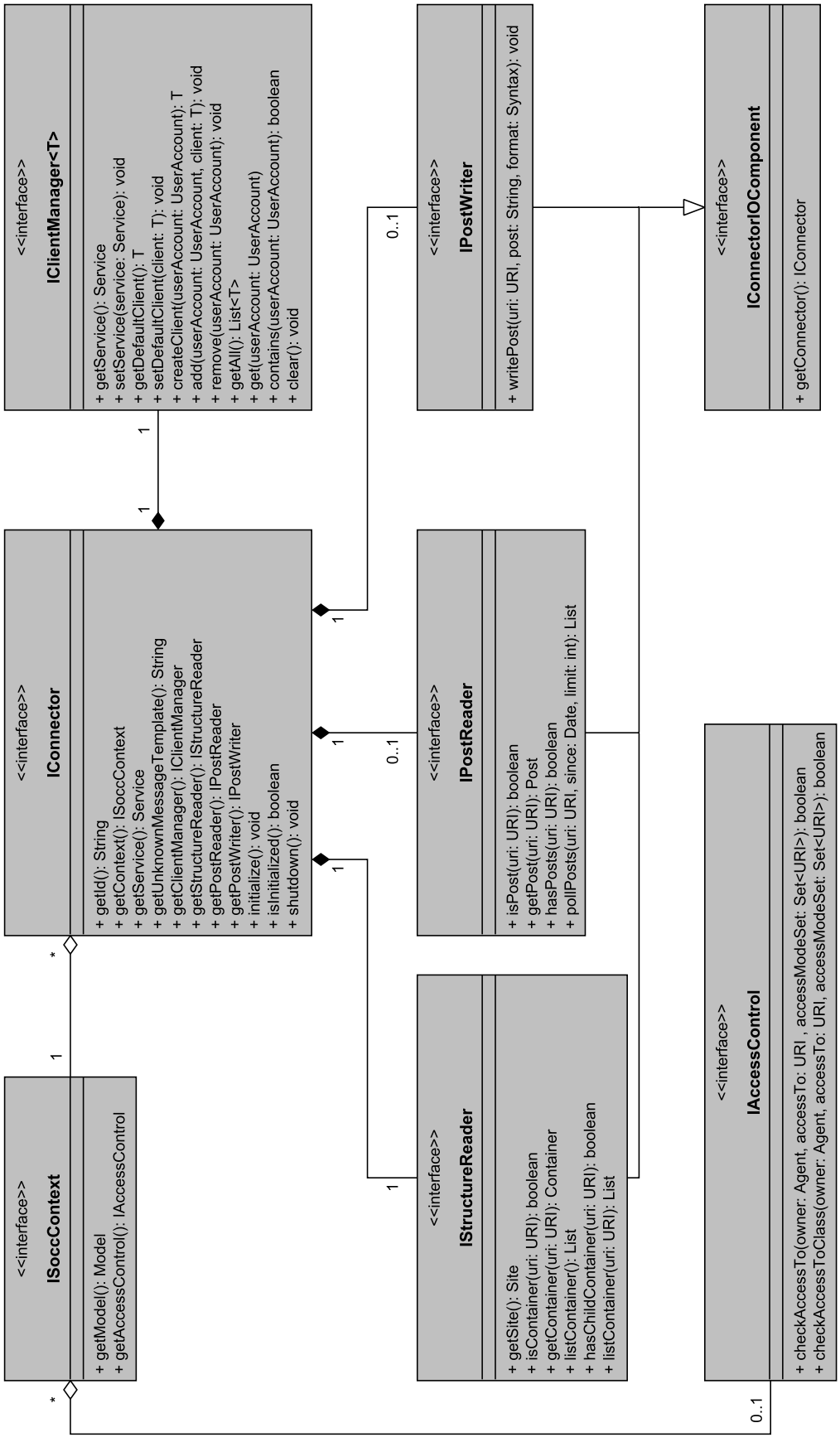


Abbildung 4.7.: UML Klassendiagramm eines Connectors

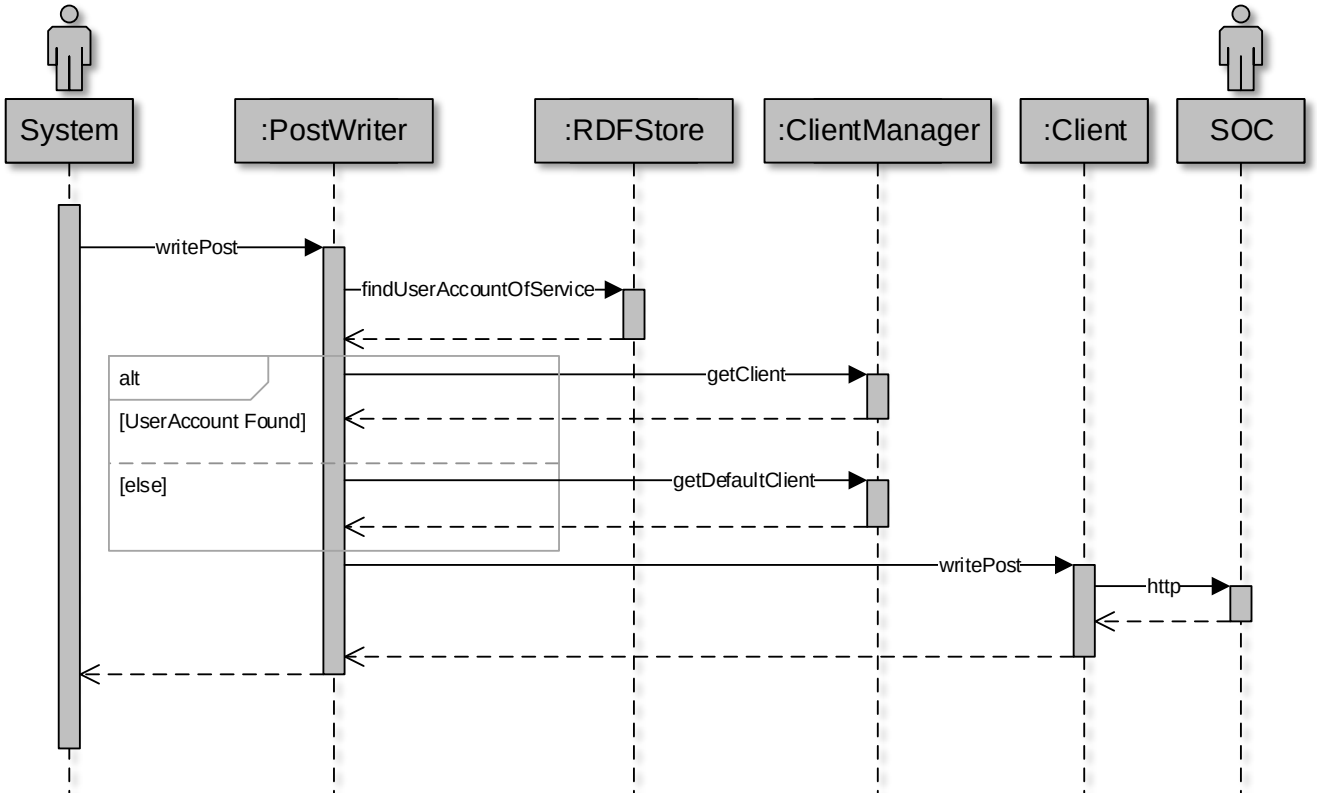


Abbildung 4.13.: UML Sequenzdiagramm eines PostWriters

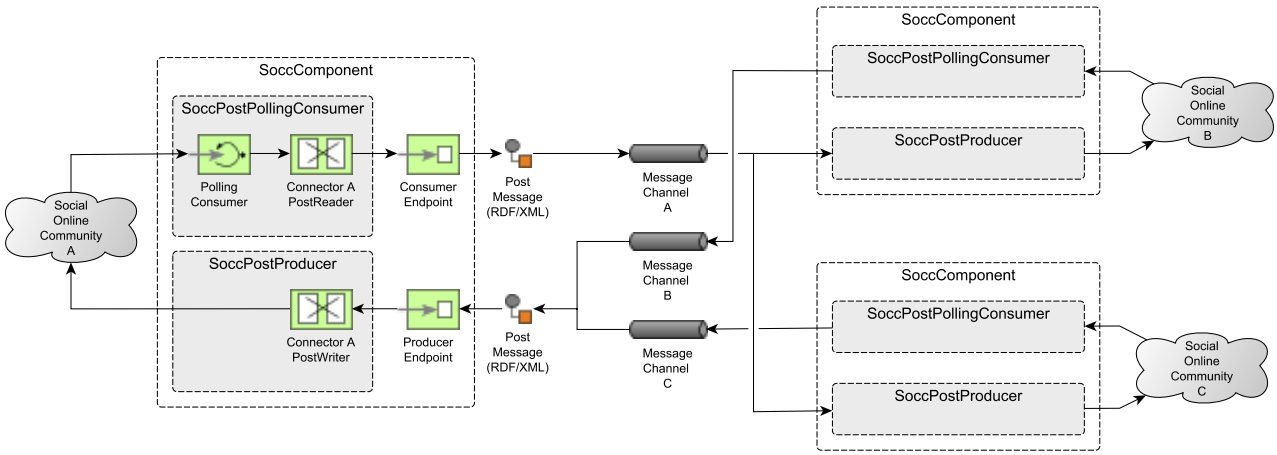


Abbildung 4.14.: Übersicht des Socc-Camel Moduls in EIP-Notation

5 Implementierung und Evaluation

5.1 Verwendete Bibliotheken

5.1.1 RDF2Go

5.2 Implementierung der Connectoren

Das muss rein:

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

Innerhalb der Mappingbeschreibungen zwischen den Format der einzelnen SOC und SIOC werden zur besseren Übersicht und einfacheren Lesbarkeit Platzhalt für einige URIs benutzt. Welche Platzhalten dies sind und für welche URI sie stehen ist aus Tabelle 5.1 zu entnehmen. Für die einzelnen SOC werden noch einmal gesonderte Platzhalter definiert, diese werden aber gesondert in den einzelnen Abschnitten beschrieben.

Tabelle 5.1.: Allgemeine Platzhalter und deren Beschreibung für das Mapping nach SIOC

Platzhalter	Bedeutung
`{serviceUri}`	URI für einen Service
`{userAccountUri}`	URI für einen UserAccount
`{siteUri}`	URI für eine Site
`{forumUri}`	URI für ein Forum
`{threadUri}`	URI für ein Thread
`{postUri}`	URI für ein Post
`{rootUri}`	Die WurzelUri einer SOC. Für Facebook wäre dies zum Beispiel `https://www.facebook.com`

5.2.1 Moodle

- 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

SIOC Mapping

API

Herausforderungen

5.2.2 Facebook

- 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

SIOC Mapping

API

Herausforderungen

5.2.3 Google+

- 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

SIOC Mapping

API

Herausforderungen

5.2.4 Youtube

- 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)

SIOC Mapping

API

Herausforderungen

5.2.5 Canvas

- 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)

SIOC Mapping

Tabelle 5.2.: Format der URIs für Canvas

URI Platzhalter	URI-Format
`{serviceUri}`	`{rootUri}/api/v1`
`{userAccountUri}`	`{rootUri}/about/{userId}`
`{siteUri}`	`{rootUri}`
`{forumUri}`	`{rootUri}/courses/{courseId}`, `{rootUri}/groups/{groupId}`
`{threadUri}`	`{forumUri}/discussion_topics/{topicId}`
`{postUri}`	`{threadUri}/entries/{entryId}`, `{threadUri}#discussion_topic`

Canvas REST-API

Der öffentliche Zugriff auf die Daten von der Lernplattform Canvas basiert auf einer REST-API und OAuth 2.0 zur Autorisierung der Zugriffe. Die Adresse, über die auf die API zugegriffen werden kann, besteht aus der URI für die verwendete Canvas Instanz (`{rootUri}`) und gefolgt von dem Pfad „/api/v1“. Für die Demo Instanz von Instructre würde dies der URI `https://canvas.instructure.com/api/v1` entsprechen. An diese können dann weitere Pfade für den Zugriff auf die einzelnen Ressourcen angehängt werden. Zur Autorisierung wird von OAuth nur der Accesstoken benötigt, der bei der REST-Abfrage als HTTP Authorization Header mitgeschickt wird. Listing 5.1 zeigt eine eine Beispielanfrage und Angabe des Accesstokens mit

dem Programm *curl*[1]. Der Platzhalter `{accessToken}` muss dann natürlich erst durch einen validen Accesstoken ersetzt werden. Diesen kann jeder Benutzter in seinem Canvas Profil unter „Settings“ und im Abschnitt „Approved Integrations“ selbst erstellen. Die Angabe von ClientId und ClientSecret von OAuth 2.0 sind nicht nötig.

Listing 5.1: Canvas Authorization Header

```
1 curl -H "Authorization: Bearer {accessToken}" https://canvas.
    instructure.com/api/v1/courses
```

Als Datenformat für die zurückgelieferten Daten wird von der Canvas API auf JSON gesetzt. Für die Verwendung von POST und PUT Operationen zum Schreiben nach Canvas können die Daten entweder nach dem HTML Form Encoding[2] Standard oder ebenfalls in JSON angegeben werden.

Da einige Anfragen eine Liste von Ergebnissen zurückliefern und diese möglicher Weise lang werden können, teilt die Canvas API diese Listen auf mehrere Seiten auf, die jede einzeln abgefragt werden müssen. Für jede Seite schickt die Canvas API mehrere URIs als HTTP Link Header[3] der Antwort mit. Diese URIs erhalten zusätzlich noch ein Attribut `rel`, das beschreibt in welcher Relation die URI zu dieser Seite steht. Als Wert für diese Relation können „current“ für eine URI auf die aktuelle, „next“ auf die nächste, „prev“ auf die vorherige, „first“ auf die erste und „last“ für eine URI auf die letzte Seite vorkommen. Um also die nächste Seite vom Ergebnis zu bekommen, muss eine neue REST-Anfrage mit der Relation „next“ ausgeführt werden. Fehlt eine URI mit dieser Relation, ist das die letzte Seite erreicht.

CanvasLMS4J

Ein Problem mit der API von Canvas war es, dass es zwar eine gute REST Anbindung gab, aber noch keine Bibliothek um sie mit der Programmiersprache Java anzusprechen. Es musste also erst eine eigne Java API dazu entwickelt werden, die den Namen *CanvasLMS4J* (Kurzform für „Canvas LMS API für Java“) bekam.

Anhand der für die REST API verwendeten URIs ist auffällig, dass die einzelnen Bestandteile aufeinander aufbauen. Zum Beispiel ist der Ablauf für den REST-Zugriff auf DiscussionTopics in Gruppen und Kursen der gleiche, nur die verwendete URI unterscheidet sich. Aus diesem Grund wurden die einzelnen Ressource (Course, Groupe, DiscussionTopic, Entries, ...) als einzelne Endpunkte implementiert die sich von der Klasse `IEnpoint` ableiten. Jeder Endpunkt kann einen Eltern-Endpunkt haben, wobei sich die endgültige URI für die REST-Abfrage aus dem Pfad des Eltern-Endpunktes und dem des aktuellen Endpunktes zusammensetzt. Zur Verdeutlichung sei hier die URI `https:{canvasUri}/api/v1/courses/1/discussion_topics` als Demonstration genannt. Sie besteht aus den statischen Teil `https:{canvasUri}/api/v1/` der den Ort für die verwendete Canvas Instanz angibt. Darauf folgt ein Kurs als erster Endpunkt mit der Kurs-ID „1“. Für diesen Kurs sollen nun alle Diskussionen abgefragt werden. Dies geschieht durch die Angabe des zweiten Endpunktes `/discussion_topics`. „/courses/1“ bildet hier also den Eltern-Endpunkt

[1] http://curl.haxx.se/
[2] http://www.w3.org/TR/html4/interact/forms.html#h-17.13.4
[3] http://www.w3.org/Protocols/9707-link-header.html

von `/discussion_topics`. Sollen aber nun alle Diskussionen in einer Gruppe abgefragt werden, reicht es aus den Kurs-Endpunkt durch einen Gruppen-Endpunkt auszutauschen.

Ausgangspunkt für CanvasLMS4J ist die Klasse `CanvasLmsClient` (siehe Abbildung 5.1). Über sie werden alle Endpunkte verwaltet die keinen Eltern-Endpunkt besitzen. Bei der Erzeugung eines Objektes dieser Klasse, werden ihr die URI zur verwendenden Canvas Instanz und der AccessToken des Benutzerkontos übergeben. Im aktuellen Stadium können von einem Client aus nur Endpunkte für Kurse oder Gruppe erstellt werden.

Endpunkte können dann über die Klasse `CanvasLmsRequest` REST-Anfragen an die Canvas API stellen. Hierzu verwendet CanvasLMS4J die *HttpClient*[4] Bibliothek von Apache mit die einzelnen HTTP-Operationen ausgeführt werden. Um die im JSON-Format zurück gelieferten Antworten der Canvas API in Java verwenden zu können, wir auf die Funktionen der von Google entwickelten *Gson*[5] Bibliothek zurück gegriffen. Sie erlaubt es Java Objekte in das JSON-Format zu konvertieren und genauso aus Daten in JSON ein Java Objekt zu machen. Dadurch verringert sich der Aufwand für das Verarbeiten der Daten von Canvas auf das Erstellen der entsprechenden Java Klassen. CanvasLmsRequest können mit zwei verschiedenen Methoden ausgeführt werden. Die erste `execute()` wird dazu verwendet, wenn die REST-Abfrage nur die Rückgabe eines Objektes zu Folge hat. Also zum Beispiel, wenn nur die Daten eines bestimmten Kurses abgefragt werden soll. Die zweite Methode ist `executePagination`. Diese Methode dient für Abfragen die eine in Seiten aufgeteilte Liste von Ergebnissen zurückliefert und gibt für eine einfache Handhabbarkeit ein Objekt der Klasse `Pagination` zurück.

Die Klasse Pagination ist nach dem Iterator Muster aufgebaut und liefert vom kompletten Ergebnis bei jedem Iterationsschritt eine Seite zurück. Diese Seite enthält dann je eine Teilliste der angefragten Objekten. Ist eine Seite fertig ausgelesen, holt Pagination über eine vorgegeben REST Abfrage die nächste Seite. Dies geschieht solange bis keine Seiten mehr geladen werden können oder das Programm das Lesen abbricht.

Listing 5.2: CanvasLMS4J Beispielprogramm

```
1  CanvasLmsClient client = new CanvasLmsClient(
2                  "https://canvas.instructure.com",
3                  "7~LUpV7B3lJY...");
4
5  Pagination<DiscussionTopic> discussionPages = client.courses()
6      .discussionTopics( 798152 )
7      .list()
8      .executePagination();
9
10
11 for ( List<DiscussionTopic> discussions : discussionPages ) {
12     for ( DiscussionTopic discussion : discussions ) {
13         System.out.println( discussion );
14     }
15 }
```

[4] http://hc.apache.org/httpclient-3.x/
[5] https://code.google.com/p/google-gson/

In Listing 5.2 ist ein Beispiel für die Anwendung der CanvasLMS4J API zu sehen. In den ersten drei Zeilen wird eine neuer CanvasLmsClient für die Canvas Instanz auf `https://canvas.instructure.com` und einem AccessToken erstellt. Als nächstes soll für einen Kurs mit der ID „798152“ alle Diskussionen aufgelistet werden. Mit dem Aufruf der Methode `courses()` des Clients wir der Endpunkt für die Kurse und von diesem aus der Endpunkt für die Diskussionen im Kurs „798152“ über die Methode `.discussionTopics( 798152 )`. In der siebten Zeile wird die Art der Abfrage genauer festgelegt. Da eine Liste alle Diskussionen gesucht ist, wird die Methode `list()` aufgerufen die ein CanvasLmsRequest Objekt für die gewünschte Abfrage erstellt. Da die Antwort aus mehrere Objekten mit der Beschreibung der einzelnen Diskussionen besteht, wird die Anfrage in Zeile Acht mit der Methode `executePagination()` ausgeführt. Die einzelnen Seiten des Ergebnisses werden dann in der elften Zeile mit einer For-Schleife durchlaufen. Das nachladen der Seiten erfolgt dabei automatisch durch die Pagination Klasse. Jede Seite besteht nun aus einer Liste mit den Beschreibung der Diskussionen in einem `DiskussionTopic` Objekt, welche dann wieder in einer weiteren Schleife ausgegeben werden.

5.3 Evaluation

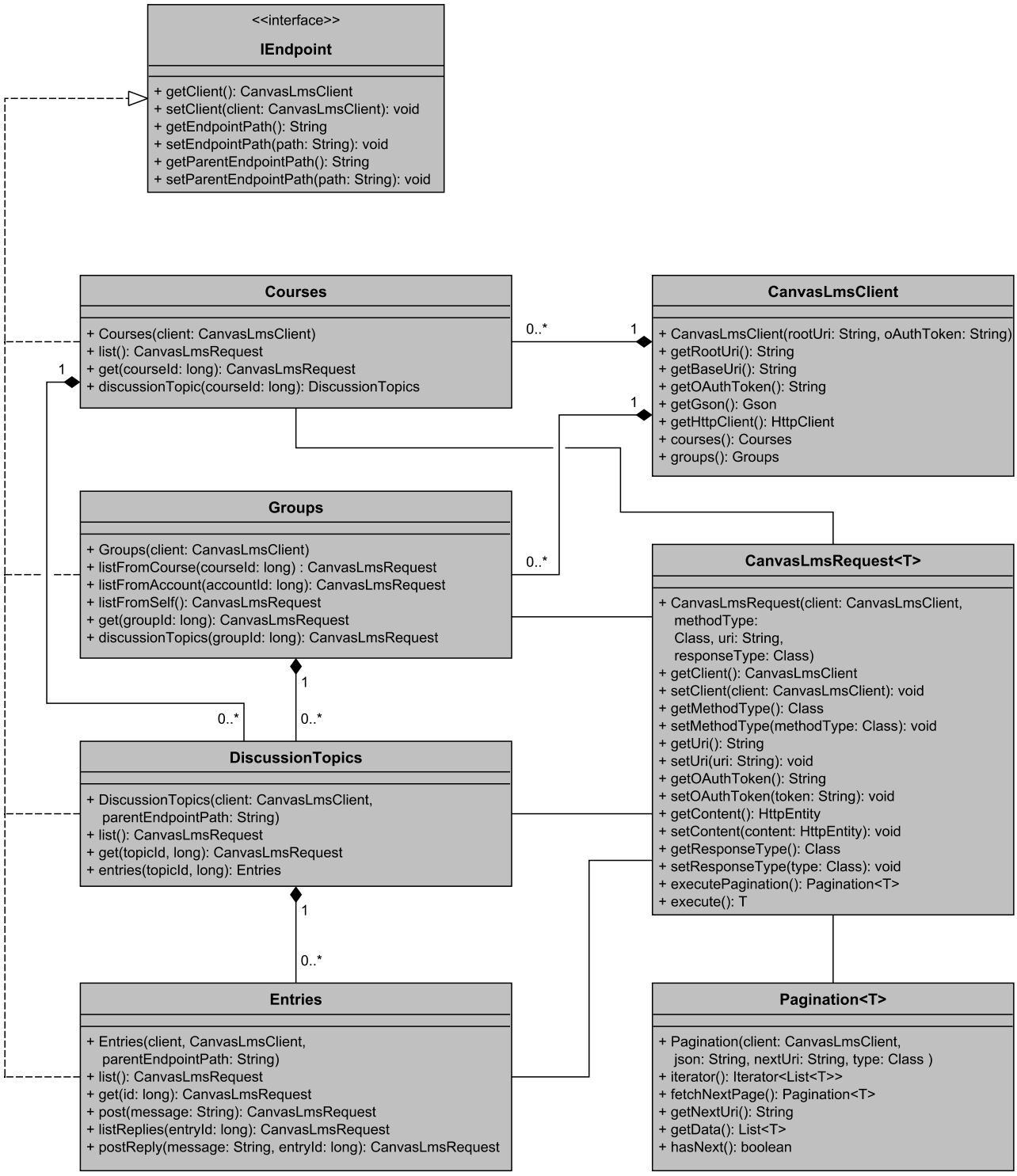


Abbildung 5.1.: UML Klassendiagramm von CanvasLMS4J

6 Abschlussbetrachtung

6.1 Fazit

6.2 Ausblick

A Anhang

A.1 SOCC Connector Config Ontologie

```
1  <?xml version="1.0"?>
2  <!--
3      Author: Florian Mueller
4      Date: 2013-09-05
5      Version: 1.2
6  -->
7
8  <!DOCTYPE rdf:RDF [
9      <!ENTITY sioc "http://rdfs.org/sioc/ns#" >
10     <!ENTITY foaf "http://xmlns.com/foaf/0.1/" >
11     <!ENTITY owl "http://www.w3.org/2002/07/owl#" >
12     <!ENTITY siocs "http://rdfs.org/sioc/services#" >
13     <!ENTITY xsd "http://www.w3.org/2001/XMLSchema#" >
14     <!ENTITY rdfs "http://www.w3.org/2000/01/rdf-schema#" >
15     <!ENTITY rdf "http://www.w3.org/1999/02/22-rdf-syntax-ns#" >
16 ]>
17
18 <rdf:RDF xmlns="http://www.m0ep.de/socc/config#"
19      xml:base="http://www.m0ep.de/socc/config#"
20      xmlns:sioc="http://rdfs.org/sioc/ns#"
21      xmlns:dcterms="http://purl.org/dc/terms/"
22      xmlns:rdfs="http://www.w3.org/2000/01/rdf-schema#"
23      xmlns:siocs="http://rdfs.org/sioc/services#"
24      xmlns:foaf="http://xmlns.com/foaf/0.1/"
25      xmlns:owl="http://www.w3.org/2002/07/owl#"
26      xmlns:xsd="http://www.w3.org/2001/XMLSchema#"
27      xmlns:rdf="http://www.w3.org/1999/02/22-rdf-syntax-ns#">
28
29     <owl:Ontology
30         rdf:about="http://www.m0ep.de/socc/config#"
31         rdf:type="http://www.w3.org/2002/07/owl#Thing">
32         <dcterms:title>SOCC Connector Configuration Ontology</
       dcterms:title>
33         <dcterms:description>
34             Ontology for connector configurations of the SOCC (
       Social Online Community Connectors) Framework.
35         </dcterms:description>
```

```
36
37          <owl:imports rdf:resource="http://rdfs.org/sioc/services#"/>
38          <owl:imports rdf:resource="http://rdfs.org/sioc/ns#"/>
39      </owl:Ontology>
40
41
42      <!--
43      ////////////////////////////
44      // Object Properties
45      ////////////////////////////
46      -->
47
48      <!-- http://www.m0ep.de/socc/config#defaultUser -->
49      <owl:ObjectProperty rdf:about="http://www.m0ep.de/socc/config#
    defaultUserAccount">
50          <rdfs:range rdf:resource="&sioc;UserAccount"/>
51          <rdfs:domain rdf:resource="http://www.m0ep.de/socc/config#
    ConnectorConfig"/>
52      </owl:ObjectProperty>
53
54      <!-- http://www.m0ep.de/socc/config#service -->
55      <owl:ObjectProperty rdf:about="http://www.m0ep.de/socc/config#
    service">
56          <rdfs:domain rdf:resource="http://www.m0ep.de/socc/config#
    ConnectorConfig"/>
57          <rdfs:range rdf:resource="&siocs;Service"/>
58      </owl:ObjectProperty>
59
60      <!--
61      ////////////////////////////
62      // Data properties
63      ////////////////////////////
64      -->
65
66
67      <!-- http://www.m0ep.de/socc/config#connectorClass Name
68          Java class of the connector
69      -->
70      <owl:DatatypeProperty rdf:about="http://www.m0ep.de/socc/config#
    connectorClassName">
71          <rdfs:domain rdf:resource="http://www.m0ep.de/socc/config#
    ConnectorConfig"/>
72          <rdfs:range rdf:resource="&xsd;string"/>
73      </owl:DatatypeProperty>
74
75      <!-- http://www.m0ep.de/socc/config#id -->
```

```
76      <owl:DatatypeProperty rdf:about="http://www.m0ep.de/socc/config#
        id">
77          <rdfs:domain rdf:resource="http://www.m0ep.de/socc/config#
        ConnectorConfig"/>
78          <rdfs:range rdf:resource="&xsd;string"/>
79      </owl:DatatypeProperty>
80
81  <!-- http://www.m0ep.de/socc/config#unknownMessageTemplate -->
82      <owl:DatatypeProperty rdf:about="http://www.m0ep.de/socc/config#
        unknownMessageTemplate">
83          <rdfs:domain rdf:resource="http://www.m0ep.de/socc/config#
        ConnectorConfig"/>
84          <rdfs:range rdf:resource="&xsd;string"/>
85      </owl:DatatypeProperty>
86
87      <!--
88      ////////////////////////////
89      // Classes
90      ////////////////////////////
91       -->
92
93      <!-- http://www.m0ep.de/socc/config#ConnectorConfig -->
94      <owl:Class rdf:about="http://www.m0ep.de/socc/config#
        ConnectorConfig"/>
95  </rdf:RDF>
96
97  <!-- Generated by the OWL API (version 3.4.2) http://owlapi.
        sourceforge.net -->
```

assets/listings/socc–config.owl

A.2 SIOC Services Authentication Module

```
1   <?xml version="1.0"?>
2   <!--
3       Author:     Florian Mueller
4       Date:       2013-08-07
5       Version:    2.0
6   -->
7
8   <!DOCTYPE RDF [
9       <!ENTITY sioc "http://rdfs.org/sioc/ns#" >
10      <!ENTITY dcterms "http://purl.org/dc/terms/" >
11      <!ENTITY owl "http://www.w3.org/2002/07/owl#" >
12      <!ENTITY sioc_services "http://rdfs.org/sioc/services#" >
```

```
13      <!ENTITY xsd "http://www.w3.org/2001/XMLSchema#" >
14      <!ENTITY rdfs "http://www.w3.org/2000/01/rdf-schema#" >
15      <!ENTITY rdf "http://www.w3.org/1999/02/22-rdf-syntax-ns#" >
16      <!ENTITY waa "http://purl.oclc.org/NET/WebApiAuthentication#" >
17  ]>
18  <rdf:RDF
19      xml:base="http://www.m0ep.de/sioc/services/auth#"
20      xmlns="http://www.m0ep.de/sioc/services/auth#"
21      xmlns:dcterms="http://purl.org/dc/terms/"
22      xmlns:owl="http://www.w3.org/2002/07/owl#"
23      xmlns:rdf="http://www.w3.org/1999/02/22-rdf-syntax-ns#"
24      xmlns:rdfs="http://www.w3.org/2000/01/rdf-schema#"
25      xmlns:sioc="http://rdfs.org/sioc/ns#"
26      xmlns:siocs="http://rdfs.org/sioc/services#"
27      xmlns:waa="http://purl.oclc.org/NET/WebApiAuthentication#"
28      xmlns:xsd="http://www.w3.org/2001/XMLSchema#">
29
30      <owl:Ontology
31          rdf:about="http://www.m0ep.de/sioc/services/auth#"
32          rdf:type="http://www.w3.org/2002/07/owl#Thing">
33          <dcterms:title>SIOC Service Authentication Module</
        dcterms:title>
34          <dcterms:description>
35              Extends the SIOC Core and Service Module to add
        information about authentication mechanisms and their required
        credentials. It reuses some parts from the WebApiAuthentication
        Ontology.
36          </dcterms:description>
37          <rdfs:seeAlso rdf:resource="http://purl.oclc.org/NET/
        WebApiAuthentication#"/>
38          <rdfs:seeAlso rdf:resource="http://rdfs.org/sioc/services#"/
        >
39
40          <owl:imports rdf:resource="http://rdfs.org/sioc/services#"/>
41      </owl:Ontology>
42
43      <!--
44      ////////////////////////////
45      // Object Properties
46      ////////////////////////////
47      -->
48
49      <!-- http://purl.oclc.org/NET/WebApiAuthentication#
        hasInputCredentials -->
50      <owl:ObjectProperty
```

```
51          rdf:about="http://purl.oclc.org/NET/WebApiAuthentication#
    hasInputCredentials">
52          <owl:equivalentProperty
53              rdf:resource="http://www.m0ep.de/sioc/services/auth#
    credentials"/>
54        </owl:ObjectProperty>
55
56        <!-- http://www.m0ep.de/sioc/services/auth#serviceAuthentication
     -->
57        <owl:ObjectProperty rdf:about="http://www.m0ep.de/sioc/services/
    auth#serviceAuthentication">
58          <rdfs:range
59              rdf:resource="http://purl.oclc.org/NET/
    WebApiAuthentication#AuthenticationMechanism"/>
60          <rdfs:domain rdf:resource="http://rdfs.org/sioc/services#
    Service"/>
61        </owl:ObjectProperty>
62
63        <!-- http://www.m0ep.de/sioc/services/auth#accountAuthentication
     -->
64        <owl:ObjectProperty rdf:about="http://www.m0ep.de/sioc/services/
    auth#accountAuthentication">
65          <rdfs:range
66              rdf:resource="http://purl.oclc.org/NET/
    WebApiAuthentication#AuthenticationMechanism"/>
67          <rdfs:domain rdf:resource="http://rdfs.org/sioc/ns#
    UserAccount"/>
68        </owl:ObjectProperty>
69
70        <!-- http://www.m0ep.de/sioc/services/auth#credentials -->
71        <owl:ObjectProperty rdf:about="http://www.m0ep.de/sioc/services/
    auth#credentials">
72          <rdfs:domain
73              rdf:resource="http://purl.oclc.org/NET/
    WebApiAuthentication#AuthenticationMechanism"/>
74          <rdfs:range rdf:resource="http://purl.oclc.org/NET/
    WebApiAuthentication#Credentials"/>
75        </owl:ObjectProperty>
76
77        <!--
78        ////////////////////////////
79        // Classes
80        ////////////////////////////
81        -->
82
83        <!-- http://purl.oclc.org/NET/WebApiAuthentication#APIKey -->
```

```
84      <owl:Class rdf:about="http://purl.oclc.org/NET/
    WebApiAuthentication#APIKey">
85          <rdfs:subClassOf rdf:resource="http://purl.oclc.org/NET/
    WebApiAuthentication#Credentials"/>
86      </owl:Class>
87
88      <!-- http://purl.oclc.org/NET/WebApiAuthentication#
    AuthenticationMechanism -->
89      <owl:Class rdf:about="http://purl.oclc.org/NET/
    WebApiAuthentication#AuthenticationMechanism"/>
90
91      <!-- http://purl.oclc.org/NET/WebApiAuthentication#Credentials
    -->
92      <owl:Class rdf:about="http://purl.oclc.org/NET/
    WebApiAuthentication#Credentials"/>
93
94      <!-- http://purl.oclc.org/NET/WebApiAuthentication#Direct -->
95      <owl:Class rdf:about="http://purl.oclc.org/NET/
    WebApiAuthentication#Direct">
96          <rdfs:subClassOf
97              rdf:resource="http://purl.oclc.org/NET/
    WebApiAuthentication#AuthenticationMechanism"/>
98      </owl:Class>
99
100     <!-- http://purl.oclc.org/NET/WebApiAuthentication#OAuth -->
101     <owl:Class rdf:about="http://purl.oclc.org/NET/
    WebApiAuthentication#OAuth">
102         <rdfs:subClassOf
103             rdf:resource="http://purl.oclc.org/NET/
    WebApiAuthentication#AuthenticationMechanism"/>
104     </owl:Class>
105
106     <!-- http://purl.oclc.org/NET/WebApiAuthentication#Password -->
107     <owl:Class rdf:about="http://purl.oclc.org/NET/
    WebApiAuthentication#Password">
108         <rdfs:subClassOf rdf:resource="http://purl.oclc.org/NET/
    WebApiAuthentication#Credentials"/>
109     </owl:Class>
110
111     <!-- http://purl.oclc.org/NET/WebApiAuthentication#Username -->
112     <owl:Class rdf:about="http://purl.oclc.org/NET/
    WebApiAuthentication#Username">
113         <rdfs:subClassOf rdf:resource="http://purl.oclc.org/NET/
    WebApiAuthentication#Credentials"/>
114     </owl:Class>
115
```

```
116     <!-- http://www.m0ep.de/sioc/services/auth#AccessToken -->
117     <owl:Class rdf:about="http://www.m0ep.de/sioc/services/auth#
    AccessToken">
118         <rdfs:subClassOf rdf:resource="http://purl.oclc.org/NET/
    WebApiAuthentication#Credentials"/>
119         <owl:equivalentClass
120             rdf:resource="http://purl.oclc.org/NET/
    WebApiAuthentication#OauthToken"/>
121     </owl:Class>
122
123     <!-- http://www.m0ep.de/sioc/services/auth#ClientId -->
124     <owl:Class rdf:about="http://www.m0ep.de/sioc/services/auth#
    ClientId">
125         <rdfs:subClassOf rdf:resource="http://purl.oclc.org/NET/
    WebApiAuthentication#Credentials"/>
126         <owl:equivalentClass
127             rdf:resource="http://purl.oclc.org/NET/
    WebApiAuthentication#OauthConsumerKey"/>
128     </owl:Class>
129
130     <!-- http://www.m0ep.de/sioc/services/auth#ClientSecret -->
131     <owl:Class rdf:about="http://www.m0ep.de/sioc/services/auth#
    ClientSecret">
132         <rdfs:subClassOf rdf:resource="http://purl.oclc.org/NET/
    WebApiAuthentication#Credentials"/>
133         <owl:equivalentClass
134             rdf:resource="http://purl.oclc.org/NET/
    WebApiAuthentication#OauthConsumerSecret"/>
135     </owl:Class>
136
137     <!-- http://www.m0ep.de/sioc/services/auth#RefreshToken -->
138     <owl:Class rdf:about="http://www.m0ep.de/sioc/services/auth#
    RefreshToken">
139         <rdfs:subClassOf rdf:resource="http://purl.oclc.org/NET/
    WebApiAuthentication#Credentials"/>
140     </owl:Class>
141
142     <!-- http://www.m0ep.de/sioc/services/auth#WebAPI -->
143     <owl:Class rdf:about="http://www.m0ep.de/sioc/services/auth#
    WebAPI">
144         <rdfs:subClassOf
145             rdf:resource="http://purl.oclc.org/NET/
    WebApiAuthentication#AuthenticationMechanism"/>
146     </owl:Class>
147 </rdf:RDF>
```

```
148 <!-- Generated by the OWL API (version 3.4.2) http://owlapi.
      sourceforge.net -->
```

assets/listings/sioc–service–auth.owl

Literaturverzeichnis

[1] WebAccessControl. http://www.w3.org/wiki/WebAccessControl, Zugriff: 2013-09-12.

[2] Apache Software Foundation. Apache Camel. http://camel.apache.org, Zugriff: 2013-09-15.

[3] David Beckett and Tim Berners-Lee. Turtle - Terse RDF Triple Language. http://www.w3.org/TeamSubmission/turtle/, Zugriff: 2013-09-13, 2011.

[4] Tim Berners-Lee. Linked Data. http://www.w3.org/DesignIssues/LinkedData.html, Zugriff: 2013-09-17, 2009.

[5] Tim Berners-Lee. Socially Aware Cloud Storage. http://www.w3.org/DesignIssues/CloudStorage.html, Zugriff: 2013-08-26, 2011.

[6] Tim Berners-Lee, James Hendler, and Ora Lassila. The semantic web. *Scientific American*, 284(5):34–43, May 2001.

[7] 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.

[8] Uldis Bojars, Alexandre Passant, John G. Breslin, and Stefan Decker. Social Network and Data Portability using Semantic Web Technologies. (1):5–19, 2008.

[9] 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.

[10] Dan Brickley and R.V. Guha. RDF Vocabulary Description Language 1.0: RDF Schema. http://www.w3.org/TR/rdf-schema/, Zugriff: 2013-09-14, 2004.

[11] Dan Brickley and Libby Miller. FOAF Vocabulary Specification 0.98. http://xmlns.com/foaf/spec/, Zugriff: 2013-09-13, 2010.

[12] Erik Christensen, Francisco Curbera, Greg Meredith, and Sanjiva Weerawarana. Web Services Description Language (WSDL). http://www.w3.org/TR/wsdl, Zugriff: 2013-09-09, 2001.

[13] Jo Davies and Martin Graff. Performance in e-learning: online participation and student grades. *British Journal of Educational Technology*, 36(4):657–663, 2005.

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

[15] Stephen Downes. E-learning 2.0. *eLearn Magazine*, 2005.

[16] Facebook. Key Facts. http://newsroom.fb.com/Key-Facts, Zugriff: 2013-09-11, 2013.

[17] Facebook in Education and Anthony Fontana. Using a Facebook Group As a Learning Management System. https://www.facebook.com/notes/facebook-in-education/using-a-facebook-group-as-a-learning-management-system/10150244221815570, Zugriff: 2013-09-14, 2010.

[18] Roy Thomas Fielding. *Architectural styles and the design of network-based software architectures*. PhD thesis, University of California, 2000.

[19] Anthony Fontana. The Multichronic Classroom : Creating an Engaging Environment for All Students. *FOUNDATIONS IN ART: THEORY AND EDUCATION, FATE IN REVIEW*, 30:13—-18, 2009.

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

[21] 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.

[22] Pascal Hitzler, Arkus Krötzsch, Sebastian Rudolph, and York Sure. *Semantic Web: Grundlagen*. Springer, 2008 edition, 2008.

[23] Gregor Hohpe and Bobby Woolf. *Enterprise Integration Patterns: Designing, Building, and Deploying Messaging Solutions*. Addison-Wesley Longman Publishing Co., Inc., Boston, MA, USA, 2003.

[24] 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.

[25] Iwen Huang. The effects of personality factors on participation in online learning. In *Proceedings of the 3rd International Conference on Ubiquitous Information Management and Communication*, ICUIMC '09, pages 150–156, New York, NY, USA, 2009. ACM.

[26] Instructure. Canvas LMS API Documentation. https://canvas.instructure.com/doc/api/index.html: Zugriff: 2013-09-19, 2013.

[27] 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.

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

[29] Frank McCown and Michael L. Nelson. What happens when facebook is gone. *Proceedings of the 2009 joint international conference on Digital libraries JCDL 09 (2009)*, pages 251–254, 2009.

[30] Nilo Mitra and Yves Lafon. SOAP Version 1.2. http://www.w3.org/TR/2007/REC-soap12-part0-20070427/, Zugriff: 2013-09-09, 2007.

[31] Stuart Palmer, Dale Holt, and Sharyn Bray. Does the discussion help? The impact of a formally assessed online discussion on final student results. *British Journal of Educational Technology*, 39(5):847–858, 2008.

[32] Peter F. Patel-Schneider, Patrick Hayes, and Ian Horrocks. OWL Web Ontology Language Semantics and Abstract Syntax. http://www.w3.org/TR/owl-semantics/, Zugriff: 2013-09-14, 2004.

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

[34] Sun/Oracle. Java Message Service. http://www.oracle.com/technetwork/java/jms/index.html, Zugriff: 2013-09-15.

[35] Watkins Thomas. Suddenly, Google Plus Is Outpacing Twitter To Become The World's Second Largest Social Network. http://www.businessinsider.com/google-plus-is-outpacing-twitter-2013-5, Zugriff: 2013-09-11, 2013.

[36] Mike Uschold and Michael Gruninger. Ontologies: Principles, methods and applications. *Knowledge engineering review*, (February), 1996.

[37] 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.

[38] Youtube. Statistik. http://www.youtube.com/yt/press/de/statistics.html, Zugriff: 2013-09-10.