Masterarbeit

Xodx

Konzeption und Implementierung eines Distributed Semantic Social Network Knotens

Natanael Arndt

28. Juni 2013

Universität Leipzig
Fakultät für Mathematik und Informatik
Institut für Informatik

Zusammenfassung

Diese Arbeit befasst sich mit dem Entwurf und der Umsetzung einer Software zum Betrieb eines Knotens in einem Distributed Semantic Social Network. Der Knoten umfasst Funktionen zur Erstellung einer persönlichen Beschreibung, zur Verwaltung von Freundschaftsbeziehungen und zur Kommunikation mit anderen Teilnehmern des Netzwerks. Die entstandene Implementierung ist bereits auf leistungsschwacher, kostengünstiger und energieeffizienter Hardware praktisch im Einsatz. Zusätzlich wurden Ihre Skalierungseigenschaften in einem Testaufbau mit mehreren Knoten untersucht.

Schlagwörter: Social Web, Semantic Web, Online Social Network, Distributed Semantic Social Network

Ich danke Sebastian Tramp für die gute Betreuung und Unterstützung während meines Studiums und der Erstellung dieser Arbeite sowie meiner Freundin Nina und meinen Kommilitonen Norman und Markus für ihre Unterstützung und guten Hinweise

Inhaltsverzeichnis

1 Einführung

Das World Wide Web (WWW) ist schon seit einiger Zeit nicht mehr nur ein System zum bloßen Abrufen von Wissen (Berners-Lee et al., 1992), sondern vielmehr ein interaktives Kommunikationsmedium. In den letzten Jahren haben sogenannte Soziale Netzwerke oder Online Social Networks an Bedeutung gewonnen. Momentan sind die wohl am meisten genutzten und als solche bekannten Online Social Networks Facebook[1] (1,1 Mrd. aktive Nutzer), Google Plus[2] (359 Mio.) und Twitter[3] (297 Mio.)[4]. Verglichen mit der geschätzten gesamten Nutzerschaft des WWW von 2,7 Mrd.[5] sind damit etwa 40 % der Nutzer des WWW bei Facebook aktiv. Diese Konzentration auf einige wenige Dienste steht im Gegensatz zur eigentlichen Organisation des WWW und des gesamten Internets als ein Netzwerk vieler dezentral organisierter Computer Knoten.

1.1 Motivation

Die momentane Situation birgt einige Risiken hinsichtlich der Privatsphäre, des Datenschutzes und der Zuverlässigkeit der Dienste. Durch den Aufbau eines verteilten Online Social Networks mit mehreren untereinander vernetzten Diensten könnten diese Risiken verringert werden. Die Wahl des Dienstes ist dem Teilnehmer dabei komplett freigestellt, einerseits kann er ein Konto bei einem größeren Anbieter anlegen und muss sich damit nicht um die Installation und den Betrieb des Dienstes kümmern, andererseits kann er aber auch soweit gehen, dass er seinen persönlichen Dienst z. B. auf einer FreedomBox (vgl. Abschn. 7.4 auf S. 59) zu Hause betreibt und damit die vollständige Kontrolle besitzt. Eine solche Topologie wäre vergleichbar mit der Verteilung von E-Mail-Servern und hätte gegenüber zentralisierten Online Social Networks Vorteile hinsichtlich Privatsphäre, Datenschutz, Urheberrecht, Erweiterbarkeit, Zuverlässigkeit und Redefreiheit (Tramp et al., 2012). Jeder Benutzer kann

[1] Facebook: http://facebook.com/

[2] Google Plus: http://plus.google.com/

[3] Twitter: http://twitter.com/

[4] Heise.de „Google+ hat 360 Millionen aktive Nutzer“: http://www.heise.de/newsticker/meldung/Google-hat-360-Millionen-aktive-Nutzer-1855169.html, 2. Mai 2013

[5] Bitkom „Das WWW wird 20 Jahre alt“: http://www.bitkom.org/de/presse/30739_76028.aspx, 30. April 2013

seine Daten, deren Verfügbarkeit und Verbreitung selbst kontrollieren ohne an die Geschäftsbedingungen eines Anbieters gebunden sein zu müssen.

Ein Vorschlag für die Architektur eines solchen verteilten Online Social Networks ist das Distributed Semantic Social Network (DSSN). Es besteht aus RDF-Ressourcen welche gemäß den Linked Data Grundsätzen (siehe Abschn. 2.1 auf S. 10) abgerufen werden können. Diese maschinenlesbaren Beschreibungen von Ressourcen, welche unter anderem die persönlichen Beschreibungen der Personen darstellen, sind auf unterschiedlichen Servern im Internet gespeichert. Jeder Knoten beinhaltet also einen Teil der Netzwerkinformationen. Teile des DSSN wurden bereits in Arndt (2010a), Tramp et al. (2011), Arndt (2011) und Weiteren beschrieben und umgesetzt, eine komplette Architektur des DSSN beschreibt „An Architecture of a Distributed Semantic Social Network“ (Tramp et al., 2012).

1.2 Ziel

Das eben beschriebene Netzwerk besteht erst in wenigen Teilen und ist für Endanwender noch nicht nutzbar. Daher soll durch diese Arbeit der Aufbau des DSSN vorangetrieben werden. Zu diesem Zweck soll ein Ansatz gefunden werden um einen Netzwek-Knoten zu entwickeln, der im DSSN gemäß der Beschreibung aus „An Architecture of a Distributed Semantic Social Network“ (Tramp et al., 2012) eingesetzt werden kann. Die entstehende Software soll die wichtigsten Funktionen zur Kommunikation in einem Online Social Network umfassen und durch zusätzliche Services erweitert werden können und mit anderen Knoten, die Teil des Netzes sind kommunizieren.

Zur Realisierung einer solchen Software zum Betrieb eines DSSN Knotens werden Anwendungsfälle und daraus resultierende Anforderungen zusammengetragen. Auf Basis der Anforderungen wird ein Entwurf der Software erarbeitet, um sie gemäß der Spezifikation umzusetzen. Zum Abschluss wird eine Evaluation der Fähigkeiten und Grenzen der entstandenen Software durchgeführt.

2 Grundlagen

Die in dieser Arbeit entwickelte Software setzt in einem großen Umfang Semantic Web-Technologien ein, aber auch Web 2.0-Technologien zur Unterstützung der Semantic Web-Technologien. Die Semantic Web-Technologien sind unter anderem Linked Data und Semantic Pingback, grundlegend für diese Techniken ist die Formulierung des Datenmodells im Resource Description Framework (RDF).

Das Resource Description Framework (RDF; Klyne & Carroll, 2004; Lassila & Swick, 1999) dient zur Beschreibung von Ressourcen. Eine Ressource kann ein Webdokument, ein Abschnitt eines Dokuments oder irgend ein anderes Ding sein, das durch einen *Uniform Resource Identifier* (URI; Berners-Lee et al., 2005) identifiziert werden kann. Um eine Ressource zu beschreiben werden ihre Eigenschaften (engl. *properties*) aufgelistet. Diese Eigenschaften können spezielle Merkmale oder Besonderheiten der Ressource aber auch Beziehungen oder Relationen zu anderen Ressourcen ausdrücken.

Die Eigenschaften werden durch Aussagen (auch *statements* oder *tripel*) formuliert. Eine Aussage setzt sich aus drei Bestandteilen zusammen, einer Ressource, deren Eigenschaft und dem entsprechenden Wert dieser Eigenschaft. Diese drei Teile werden auch Subjekt, Prädikat und Objekt genannt. Das Objekt kann entweder eine Ressource mit URI oder ein Literal sein, dies kann bei der Definition der Eigenschaft, durch die Verwendung der Klasse `owl:DatatypeProperty` oder `owl:ObjectProperty`, festgelegt werden. Durch die Angabe einer `rdfs:domain` oder `rdfs:range` kann zudem der Typ des Subjekts resp. des Objekts festgelegt werden.

Da durch eine Wiederverwendung der Ressourcen und Eigenschaften in mehreren Beschreibungen die Verknüpfung der einzelnen Datensätze untereinander gestärkt wird und somit die semantische Auswertung vereinfacht wird, werden Ressourcen und Eigenschaften in Vokabularen zusammengefasst. Diese Vokabulare beinhalten außerdem eine Beschreibung der Semantik der Ressourcen und Eigenschaften. Ressourcen und Eigenschaften eines Vokabulars sind meist in einem gemeinsamen Namensraum definiert, d. h. die URIs beginnen gleich. Der gleich bleibende Teil des URI wird in der Regel durch ein Präfix (engl. *prefix*) abgekürzt. Die in dieser Arbeit verwendeten Präfixe sind in Lst. 5.1 auf S. 32 definiert.

Grundlegend für RDF ist das Konzept des URI, dieses ist in RDF kompatibel zu Internationalized Resource Identifiers (IRIs; Klyne & Carroll, 2004, 6.4 „RDF URI References“), daher werden IRI und URI hier gleichgesetzt und einheitlich URI genannt.

2.1 Linked Data

Linked Data ist das Prinzip der Erreichbarkeit von semantischen Webinhalten, indem man den URI im Internet nachschlägt. Tim Berners-Lee fasst dazu in seinen „Linked Data – Design Issues“ (Berners-Lee, 2009) folgende vier Regeln zusammen:

1. Use URIs as names for things
2. Use HTTP URIs so that people can look up those names.
3. When someone looks up a URI, provide useful information, using the standards (RDF*, SPARQL)
4. Include links to other URIs. so that they can discover more things.

Solange URIs vergeben werden, über deren Namensraum man selbst entscheiden kann, können Ressourcen weltweit eindeutig identifiziert werden (Punkt 1), ohne Namenskonflikte zu befürchten. Indem URIs mit dem HTTP-Schema verwendet werden, können sie über einen weit verbreiteten Standard angefragt werden (Punkt 2). Auf eine Anfrage dieses URI sollte mit strukturierten Daten geantwortet werden. Diese Daten sollten in einem maschinenlesbaren und möglichst weit verbreiteten Format formuliert sein (Punkt 3). Um die abgefragte Ressource in einen größeren Kontext zu setzen, und dadurch einen Hinweis auf weiterführende Informationen zu geben, sollte auf andere Ressourcen verwiesen werden (Punkt 4). Diese Regeln sind keine Vorschriften, sondern Vorschläge. Durch ihr Einhalten ergeben sich sonst nicht vorhandene neue Möglichkeiten Daten gemeinsam zu nutzen. Da diese Vorgehensweise im Semantic Web sehr weit verbreitet ist, wird sie in dieser Arbeit genutzt.

2.2 Semantic Pingback

Semantic Pingback ist eine Erweiterung des Pingback-Protokolls (Langridge & Hickson, 2002). Pingback wird zum Aufbau von Rückverweisen (*back link* oder *trackback*) zwischen Blogs eingesetzt, dazu sendet ein Blog einen Ping an einen von ihm verlinkten Blog. Semantic Pingback überträgt dieses Prinzip auf RDF-Ressourcen und wurde in „Weaving a Social Data Web with Semantic Pingback“ (Tramp et al., 2010) beschrieben. Es ist ein proaktives Benachrichtigungssystem und ist abwärts-

kompatibel zu Pingback. Zusätzlich zu den XML-RPCs (*XML remote procedure call*) spezifiziert Semantic Pingback einen Ping als einfache HTTP-POST-Anfrage.

Die HTTP-POST-Anfrage ist so spezifiziert, dass eine Ressource (*target*) über eine Erwähnung in einer Beschreibung einer anderen Ressource (*source*) informiert wird. Abbildung 2.1 zeigt einen möglichen Ablauf des Semantic Pingback-Protokolls. Das *target* muss mit der Relation `pingback:to` einen Pingback-Service angeben, um Pings empfangen zu können. Der Ping wird dann an den Pingback-Service, der im *target* angegeben ist, gesendet. Kommt bei dem Pingback-Service ein Ping an, überprüft er, ob die *source* tatsächlich eine Erwähnung des *target* oder einen Verweis auf das *target* enthält. War die Prüfung des Pings erfolgreich und somit der Ping valide, reagiert der Pingback-Service auf eine bei sich festgelegte Weise. Dies kann z. B. geschehen, indem der ebenfalls einen Verweis auf die *source* zur Beschreibung des *target* hinzufügt und den Eigentümer des *target* darüber informiert.

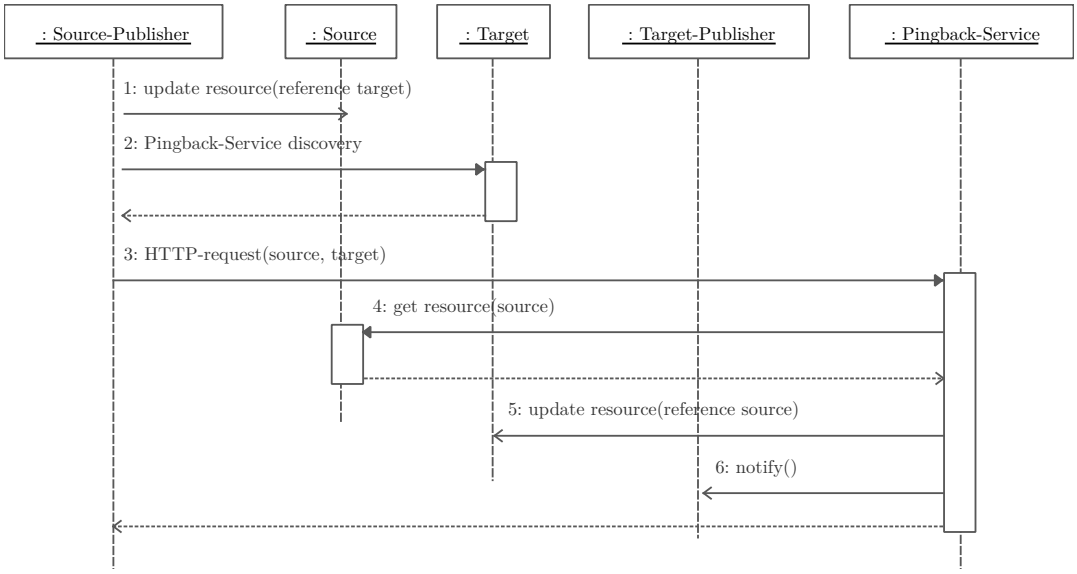


Abbildung 2.1: Sequenzdiagramm des Ablaufs eines Semantic Pingback-Pings

2.3 PubSubHubbub

Das PubSubHubbub-Protokoll ist ein *publish/subscribe* Protokoll für Atom- und RSS-Feeds das mit *web-hooks* arbeitet[1]. Das bedeutet, dass ein Abonnent (*subscriber*) Aktualisierungen zu einem Feed abonniert und der Herausgeber (*publisher*) ihn mit Hilfe eines *web-hook* darüber informiert, sobald der Feed aktualisiert wird.

[1] PubSubHubbub: `http://pubsubhubbub.appspot.com/`, `https://code.google.com/p/pubsubhubbub/`

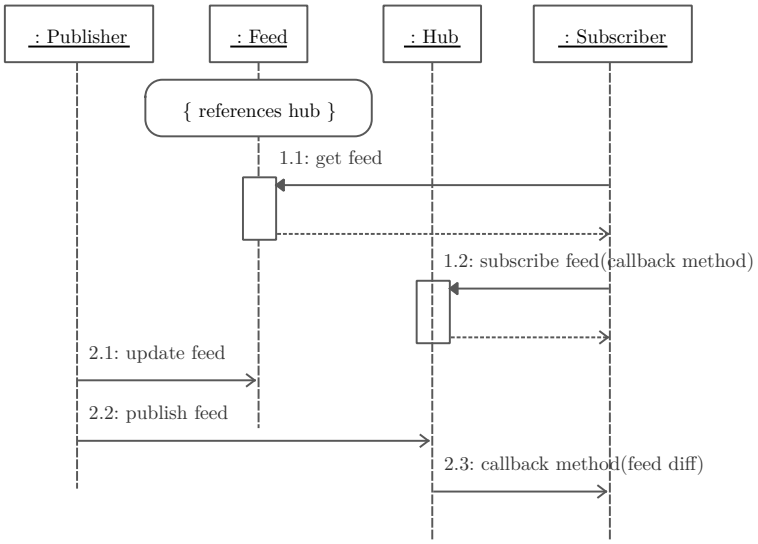


Abbildung 2.2: Sequenzdiagramm zur Veranschaulichung eines *subscribe*- (1.x) und eines *publish*-Ablaufes (2.x). Der *subscriber* steht dabei beispielhaft für mehrere Abonnenten.

Um die Last auf den *publisher* zu verringern, wird zusätzlich ein *hub* eingeführt. Der *hub* wird vom *publisher* ausgesucht und im Kopf des Feeds angegeben. Der Ablauf des PubSubHubbub-Protokolls wird in Abb. 2.2 dargestellt. Möchte ein *subscriber* den Feed abonnieren liest er diese Angabe beim ersten Aufruf des Feeds (1.1) und sendet anschließend eine HTTP-Anfrage unter Angabe einer *call back*-Funktion an den *hub* (1.2). Wenn der *publisher* den Feed aktualisiert (2.1) sendet er ebenfalls eine HTTP-Anfrage an den *hub* (2.2), welcher daraufhin die *subscriber* über deren *call back*-Funktionen (*web-hook*) benachrichtigt (2.3).

2.4 Activity-Streams

Activity-Streams[2] sind zeitlich geordnete Sammlungen von Aktivitäten von Personen in Social Networks. Sie können sowohl in Form von Atom-Feeds (Nottingham & Sayre, 2005) mit dem XML-Namespace `http://activitystrea.ms/schema/1.0/` (Hollander et al., 1999), als auch als JSON-Dokument, formuliert werden.

Der grundlegende Aufbau der Aktivitäten ist im „Activity Base Schema“ (Snell et al., 2012) formuliert. Die Datenstrukturen der JSON-Dokumente werden in „Json Activity Streams 1.0“ (Snell et al., 2011) beschriebn. Für diese Arbeit sind allerdings nur

[2]Activity-Streams: `http://activitystrea.ms/`

die Spezifikationen für Atom-Feeds relevant, welche in „Atom Activity Streams 1.0“ (Atkins et al., 2011) beschrieben werden.

Eine Aktivität (*activity*) ist ein Atom-Entry und besitzt zusätzlich die Angaben des Akteurs, Verbs und Objekts. Der Akteur ist die Angabe der handelnden Person in Form eines `atom:author`. Das Verb beschreibt die durchgeführte Handlung. Als allgemeines Verb wurde `post` spezifiziert, alle Verben aus dem Activity-Streams-Namespace können unter weglassen des Namespaces abgekürzt werden. Weitere Verben sind z. B. `like`, `make-friend` und `share`, es können aber auch eigene Verben verwendet werden. Das Objekt beschreibt den Gegenstand der Handlung. Der Typ der Handlung wird durch die Angabe des `activity:object-type` festgelegt. Ebenso wie Verben, können auch Angaben des Object-Types abgekürzt werden. Es gibt eine Reihe vordefinierter Object-Types z. B. `bookmark`, `comment` und `note`, es können aber auch hier eigene Typen definiert werden.

RDF Mapping

Um die Aktivitäten, die intern verwaltet und von eingehenden Atom-Feeds gewonnen werden zu beschreiben, wird das „Atom Activity Streams RDF mapping“ (Minno & Palmisano, 2010) eingesetzt. Es beschreibt eine direkte Abbildung des Activity-Streams-Namespaces auf RDF-Ressourcen und -Eigenschaften im RDF-Namespace `aair`.

Abweichend von der Spezifikation im `aair`-Vokabular wird als Objekt der Eigenschaft (*range*) `aair:activityActor` eine Ressource vom Typ `foaf:Person` verwendet. Das Vokabular gibt als *range* der Relation `aair:Actor` an, in der Spezifikation des Atom Activity-Streams wird als Actor ein `atom:author`-Element spezifiziert. In Nottingham & Sayre (2005) steht „The ”atom:author” element is a *Person* construct that indicates the author of the entry or feed.“[3]. Daher ist die Verwendung von `foaf:Person` zur Angabe des `aair:activityActor` angebracht.[4]

[3] RFC 4387 Abschnitt 4.2.1: `http://tools.ietf.org/html/rfc4287#section-4.2.1` (Nottingham & Sayre, 2005)

[4] Dies wurde auch auf der Mailingliste zum notube-Projekt angemerkt: `http://mailman.few.vu.nl/pipermail/public-notube/2013-May/000179.html`

3 Anforderungen

Im Rahmen dieser Arbeit soll ein Knoten eines Online Social Networks entworfen und implementiert werden. Zu dem Zweck werden in diesem Kapitel einige Anforderungen an das entstehende System definiert. Über die Zeit haben sich bestimmte Funktionalitäten herausgebildet, die von Teilnehmern an Online Social Networks häufig zur Kommunikation mit anderen Teilnehmern verwendet werden. Diese Funktionen, aber auch andere, noch nicht so weit verbreitete Anforderungen, sollen von dem hier entworfenen Netzwerk zur Verfügung gestellt werden. Nach der Beschreibung der beteiligten Rollen werden Anwendungsfälle (AF) beschrieben, aus denen anschließend die Benutzeranforderungen (BA) extrahiert werden.

3.1 Rollen

Zur Beschreibung der Anforderungen werden unterschiedliche Rollen verwendet. Um ein einheitliches Verständnis dieser Rollen zu gewährleisten werden sie im Folgenden beschrieben. Zusätzlich zu den beschriebenen Rollen wird in den nächsten Abschnitten bereits teilweise die in Abschn. 5.1 auf S. 31 definierte Terminologie verwendet.

Person ist eine natürliche Person. Sie kann an dem Online Social Network teilnehmen und es zur Kommunikation mit anderen Personen nutzen. (vgl. `foaf:Person`)

Freund ist eine Person, die in einer Bekanntschaftsbeziehung mit einer anderen Person steht. (Z. B.: Amélie kennt Nino, also ist Nino für Amélie ein Freund)

Agent ist eine Person, eine Organisation, eine Gruppe, ein Gerät, ein Computerprogramm oder irgendeine andere handelnde Einheit (vgl. `foaf:Agent`). Er kann Aktionen im Online Social Network durchführen und handelt unter Umständen im Auftrag einer Person. Eine Person ist auch ein Agent.

(DSSN-)Knoten ist ein Netzwerkknoten eines verteilten Online Social Networks bzw. des DSSN. Die Knoten unterteilen sich abhängig von ihren Aufgaben im Netzwerk bzw. ihrer Position in der Ebenen-Architektur (siehe Abschn. 5.2.1 auf S. 33) in Daten-Knoten, Protokoll-Knoten, Service-Knoten und Anwen-

dungs-Knoten. Ein Knoten kann aber auch mehrere Aufgaben in unterschiedlichen Ebenen erfüllen. Jeder Knoten stellt so einen Teil der Funktionen des Netzwerks bzw. der DSSN Architektur zur Verfügung.

3.2 Anwendungsfälle

Die folgenden Anwendungsfälle stellen beispielhaft und abstrakt die Nutzung eines Online Social Networks dar und sollen dabei für mehrere Möglichkeiten zur Benutzung des Systems in einer realen Umgebung stehen. Aus jedem Anwendungsfall werden die Akteure und deren Interaktionen extrahiert. Von den Interaktionen und Funktionen wird auf die entsprechenden Benutzeranforderungen verwiesen.

3.2.1 Foto teilen, taggen und kommentieren

/AF1/

Im Social Web Acid Test – Level 0 (SWAT0)[1] wird ein Anwendungsfall beschrieben, der das Hochladen, Taggen und Kommentieren eines Fotos beinhaltet. (Siehe dazu auch Abschn. 7.2 auf S. 57.) Abbildung 3.1 auf S. 16 stellt den Anwendungsfall in einem Sequenzdiagramm mit der Zusatzforderung dar, dass jede Person einen eigenen Server nutzt.

Dave fotografiert mit seinem Telefon Tantek, tagged das Foto mit Tantek, und lädt es auf seinen Server hoch. Tantek erhält eine Benachrichtigung, auf einem anderen Server darüber, dass er auf dem Foto getagged wurde. Evan, der Dave folgt (sein Profil abonniert hat), sieht das Foto auf seinem Server. Evan kommentiert das Foto. Dave und Tantek erhalten eine Benachrichtigung, dass Evan das Foto kommentiert hat.

Akteure mit Rollen: Dave (Person), Daves Server (Knoten), Tantek (Person), Tanteks Server (Knoten), Evan (Agent oder Person) und Evans Server (Knoten)

Aktionen und Anforderungen:

Kommunikation über Server- bzw. Anbietergrenzen hinweg (BA 3 auf S. 21)

Persönliche Beschreibung anlegen und bearbeiten (BA 4 auf S. 22)

Medien-Artefakte veröffentlichen und Bilder taggen (BA 8 auf S. 22)

Ressourcen abonnieren und Abonnementverwaltung (BA 10 auf S. 23)

[1] Social Web Acid Test – Level 0: `http://www.w3.org/2005/Incubator/federatedsocialweb/wiki/SWAT0/`

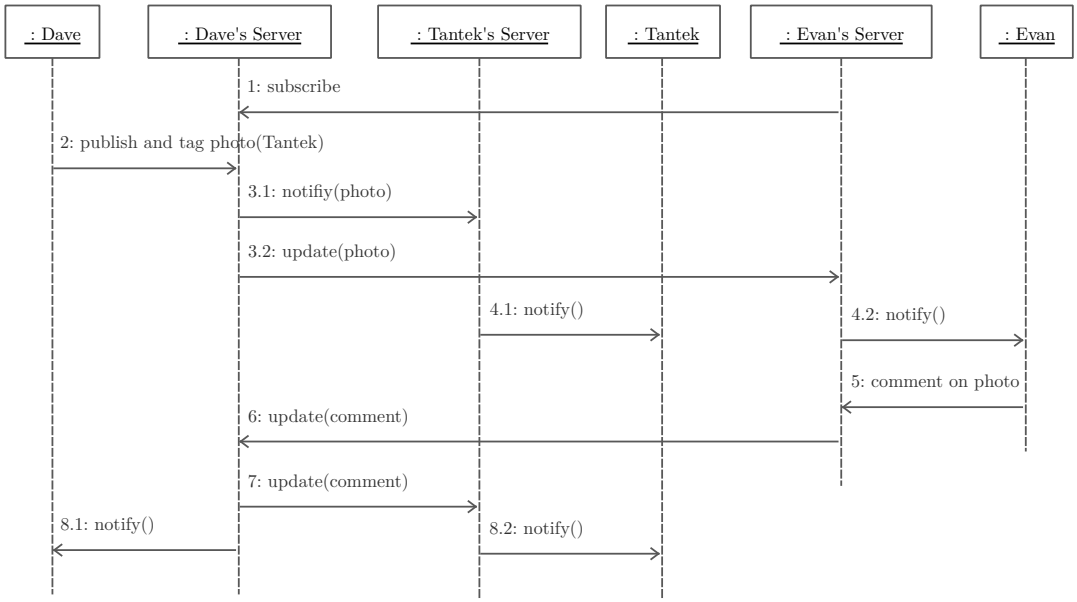


Abbildung 3.1: Sequenzdiagramm des Anwendungsfalls „Foto teilen, taggen und kommentieren“ mit den Interaktionen der einzelnen Akteure

Aktualisierungen zu abonnierten Ressourcen empfangen (BA 11 auf S. 23)

Ressourcen kommentieren (BA 12 auf S. 23)

Benachrichtigungen (BA 13 auf S. 23)

3.2.2 Soziale Kommunikation und Freundschaft schließen

/AF2/

Amélie möchte gerne an der fabelhaften Welt der Sozialen Netzwerke teilnehmen. Sie legt sich ein Profil an, um darauf über Erlebnisse zu berichten. Später lernt Amélie Nino kennen und möchte mit ihm in Kontakt treten. Sie betrachtet sein Profil, die von ihm geteilten Fotos und seine Statusnachrichten und möchte auch in Zukunft Neuigkeiten von ihm erfahren.

Abb. 3.2 auf S. 17 stellt einen möglichen Ablauf der Handlungen dieses Anwendungsfalls in einem Sequenzdiagramm dar. Dabei wird pro Person ein eigener Server modelliert. Diese Konfiguration geht nicht direkt aus dem Anwendungsfall hervor und ist auch nicht zwingend notwendig, vielmehr soll sie die Möglichkeit der Verteilung hervorheben. Die beiden Server können aber auch als ein gemeinsamer betrachtet werden, bei dem die *notify and subscribe* und *update* Nachrichten intern übermittelt werden.

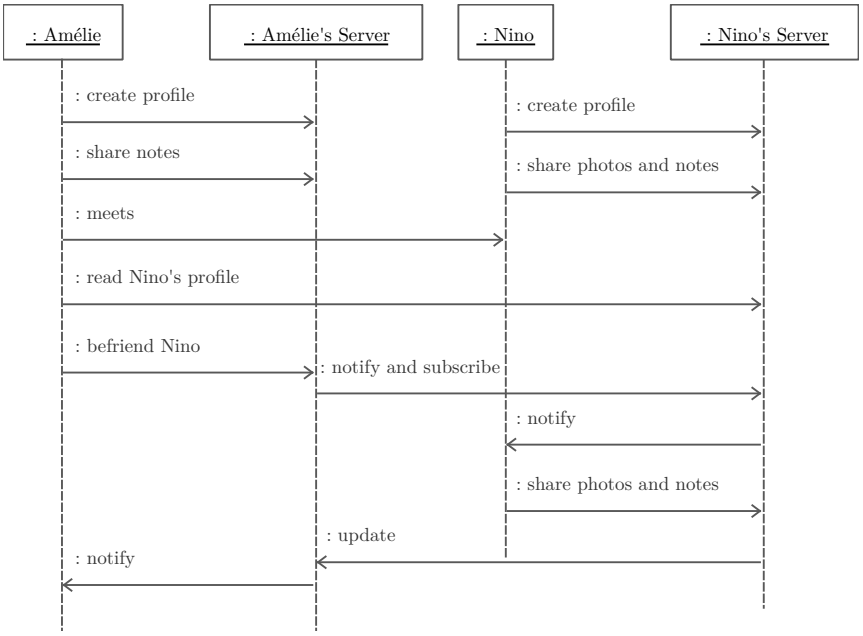


Abbildung 3.2: Sequenzdiagramm des Anwendungsfalls „Soziale Kommunikation und Freundschaft schließen“ mit den Interaktionen der einzelnen Akteure

Akteure mit Rollen: Amélie (Person), Amélies Server (Knoten), Nino (Freund) und Ninos Server (Knoten)

Aktionen und Anforderungen:

Persönliche Beschreibung anlegen und bearbeiten (BA 4 auf S. 22)

Statusnachrichten veröffentlichen (BA 7 auf S. 22)

Freunde hinzufügen (BA 6 auf S. 22)

Beschreibungen anderer Personen abrufen (BA 5 auf S. 22)

Medien-Artefakte veröffentlichen und Bilder taggen (BA 8 auf S. 22)

Ressourcen abonnieren und Abonnementverwaltung (BA 10 auf S. 23)

Aktualisierungen zu abonnierten Ressourcen empfangen (BA 11 auf S. 23)

Benachrichtigungen (BA 13 auf S. 23)

3.2.3 Investigativer Journalismus

/AF3/

Peter ist Journalist und arbeitet zusammen mit seinem Kollegen Paul an verschiedenen investigativen Berichten. Er ist besorgt, dass Daten über Informanten und brisante Details in die falschen Hände geraten. Daher möchten Peter und Paul gerne die Daten jeweils bei sich zu Hause bzw. im Büro speichern und kontrollieren wer sie lesen darf. Um dies sicherzustellen beschaffen sich Peter und Paul je einen kleinen Server (z. B. eine FreedomBox), diese Server möchten sie zur Kommunikation nutzen. Da die Stromversorgung zu Hause nicht immer sichergestellt ist, richten sie zusätzlich einen Server in der Redaktion ein, über den sie bei einem Stromausfall kommunizieren können.

In Abb. 3.3 wird der Aufbau der Knoten und der mögliche Nachrichtenfluss zwischen den Knoten in Form eines Kommunikationsdiagramms dargestellt. Es müssen aber nicht immer über alle Verbindungen Nachrichten gesendet werden.

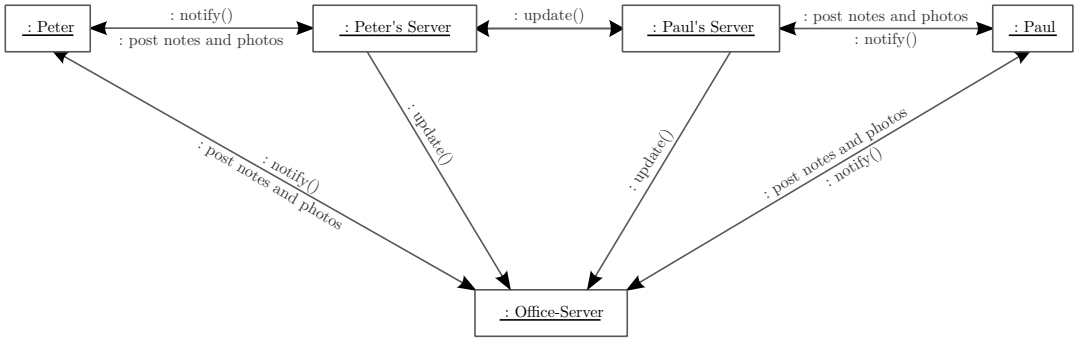


Abbildung 3.3: Kommunikationsdiagramm zur Darstellung des Anwendungsfalls „Investigativer Journalismus“ mit dem Datenfluss zwischen den einzelnen Akteuren

Akteure mit Rollen: Peter (Person), Peters Server (Knoten), Paul (Freund oder Agent), Pauls Server (Knoten) und Büro-Server (Knoten)

Aktionen und Anforderungen:

Privatsphäre, Datenschutz und Redefreiheit (BA 1 auf S. 20)

Zuverlässigkeit (BA 2 auf S. 21)

Kommunikation über Server- bzw. Anbietergrenzen hinweg (BA 3 auf S. 21)

Statusnachrichten veröffentlichen (BA 7 auf S. 22)

Medien-Artefakte veröffentlichen und Bilder taggen (BA 8 auf S. 22)

Beliebige Internetseiten und Ressourcen teilen (BA 9 auf S. 23)

Ressourcen kommentieren (BA 12 auf S. 23)

3.2.4 Event organisieren

/AF4/

DJ Tectonics möchte ein Event „Plattenverschiebung“ organisieren und über seine Profilseite bei seinen Fans bewerben. Um ungefähr abschätzen zu können, wie viel Bier er besorgen muss, möchte er seinen Fans die Möglichkeit einer Rückmeldung („Ich komme“, „Ich komme nicht“ oder „Ich komme vielleicht“) geben. DJ Tectonics legt auf einem Kalender-Service eine Event-Ressource an, diese wird automatisch in seinen Activity-Stream auf seinem Server eingetragen, wodurch alle Fans (Abonnenten) darüber benachrichtigt werden, darüber diskutieren und eine Rückmeldung geben können. Den Fans gefällt das und sie fügen daher einen Verweis (*like*) auf das Event zu ihren persönlichen Beschreibungen hinzu.

Abbildung 3.4 zeigt einen möglichen Ablauf zur Organisation eines Events. Dabei wird ein externer Service eingesetzt, der sich im DSSN integriert und im Namen von DJ Tectonics auf dessen Knoten eine neue *activity* anlegt.

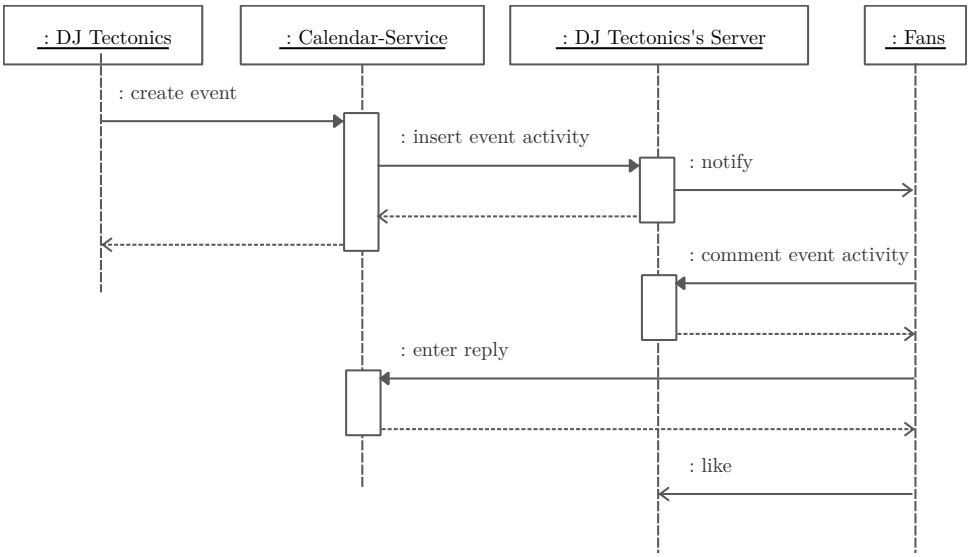


Abbildung 3.4: Sequenzdiagramm zur Darstellung des Anwendungsfalls „Event organisieren“ mit den Interaktionen der einzelnen Akteure

Akteure mit Rollen: DJ Tectonics (Person), Kalender-Service (Agent), DJ Tectonics Server (Knoten) und Fans (Personen)

Aktionen und Anforderungen:

Kommunikation über Server- bzw. Anbietergrenzen hinweg (BA 3 auf S. 21)

Persönliche Beschreibung anlegen und bearbeiten (BA 4 auf S. 22)

Beschreibungen anderer Personen abrufen (BA 5 auf S. 22)

Freunde hinzufügen (BA 6 auf S. 22)

Statusnachrichten veröffentlichen (BA 7 auf S. 22)

Beliebige Internetseiten und Ressourcen teilen (BA 9 auf S. 23)

Ressourcen abonnieren und Abonnementverwaltung (BA 10 auf S. 23)

Aktualisierungen zu abonnierten Ressourcen empfangen (BA 11 auf S. 23)

Ressourcen kommentieren (BA 12 auf S. 23)

Benachrichtigungen (BA 13 auf S. 23)

3.3 Benutzeranforderungen

> „User requirements are statements, in a natural language plus diagrams, of what services the system is expected to provide to system users and the constraints under which it must operate.“ (Sommerville, 2011, S. 83)

Aus den obigen Anwendungsfällen ergeben sich die folgenden Benutzeranforderungen an das Online Social Network. In einem verteilten Online Social Network können diese Funktionen auch von mehreren Knoten gemeinsam oder unabhängig voneinander zur Verfügung gestellt werden. Um den Aufbau des gesamten Netzwerks zu vereinfachen, soll der in dieser Arbeit entworfene Knoten möglichst viele grundlegende Funktionen vereinen. Es lassen sich also die folgenden Anforderungen an ein Online Social Network direkt auf die entstehende Knoten-Implementierung übertragen.

3.3.1 Privatsphäre, Datenschutz und Redefreiheit

/BA1/

Online Social Networks werden meist zu einem hohen Maß für private und zum Teil auch vertrauliche Kommunikation zwischen Personen verwendet, die gespeicherten Daten bedürfen daher eines besonderen Schutzes. In herkömmlichen, von einer zentralen Instanz betriebenen Online Social Networks ist dies nicht gegeben und hat bereits zu Problemen geführt. Im April 2011 wurde zum Beispiel das Sony Playstation Network angegriffen und Profildaten gestohlen[2], sodass das Netzwerk anschließend

[2] Artikel auf heise.de zu dem Angriff aus das Playstation-Network: `http://www.heise.de/newsticker/meldung/Angriff-auf-Playstation-Network-Persoenliche-Daten-von-Millionen-Kunden-gestohlen-1233136.html`, 27. April 2011; Stellungnahme von Sony zu den Vorfällen: `http://blog.de.playstation.com/2011/04/26/psnqriocity-service-update/`, 26. April 2011

längere Zeit nicht erreichbar war[3]. In China kommt es im Zusammenhang mit Twitter und Flickr zu Einschränkungen der Redefreiheit[4]. Das hier entstehende Online Social Network soll daher den teilnehmenden Personen ermöglichen selbst die Verbreitung und Speicherung ihrer Daten zu kontrollieren. Durch die Vermeidung eines einzigen Angriffspunktes (*single point of attack*) sollen der Datenschutz und die Redefreiheit erhöht werden.

3.3.2 Zuverlässigkeit

/BA2/

Ein Online Social Network sollte möglichst ausfallsicher sein um stets die Kommunikation zwischen den teilnehmenden Personen zu ermöglichen. Facebook kann diese Zuverlässigkeit nicht immer garantieren, so ist zum Beispiel das Netzwerk im März 2012 in Europa, Afrika und im Nahen Osten aufgrund eines DNS-Fehlers für einige Stunden nicht erreichbar gewesen[5]. Durch das Vermeiden eines solchen einzigen Fehlerpunktes (*single point of failure*) soll daher die Zuverlässigkeit erhöht werden (Dooley, 2001).

3.3.3 Kommunikation über Server- bzw. Anbietergrenzen hinweg

/BA3/

Es soll möglich sein mit anderen Personen zu kommunizieren, unabhängig davon, auf welchem Server bzw. bei welchem Anbieter eine Person ihr Profil angelegt hat. (Dies ist vergleichbar mit der Kommunikation über E-Mail.) Ebenso soll es möglich sein, mit seinen Daten von einem Server bzw. Anbieter zu einem anderen umzuziehen. Dabei ist es wichtig, dass die Daten möglichst unabhängig von der konkreten Anwendung formuliert sind, um sie auch mit anderen Implementierungen nutzen zu können und damit die Erweiterbarkeit des gesamten Netzes sicherzustellen.

[3] heise.de – „Sony will abgeschaltete Online-Dienste in Kürze wieder anbieten“: `http://www.heise.de/newsticker/meldung/Sony-will-abgeschaltete-Online-Dienste-in-Kuerze-wieder-anbieten-1235320.html`, 1. Mai 2011

[4] USA today – „Social-networking sites Twitter, Flickr go dark in China“: `http://usatoday30.usatoday.com/tech/news/2009-06-02-china-twitter-tiananmen-protests_N.htm`, 6. März 2009; Reporter ohne Grenzen – Enemies of the Internet: `http://en.rsf.org/IMG/pdf/Internet_enemies.pdf`, 12. März 2010

[5] ZDNet – „Facebook is down in Europe“: `http://www.zdnet.com/blog/facebook/facebook-is-down-in-europe/10080/`, 6. März 2012

3.3.4 Persönliche Beschreibung anlegen und bearbeiten

/BA4/

Um an einem Online Social Network teilzunehmen, muss eine Person eine persönliche Beschreibung anlegen können. Diese Seite dient für andere Teilnehmer zur Identifikation der Person und beinhaltet Verweise auf deren Freunde. Eine bereits bestehende persönliche Beschreibung soll importiert und weiterverwendet werden können.

3.3.5 Beschreibungen anderer Personen abrufen

/BA5/

Es soll Teilnehmern des Online Social Networks möglich sein, Beschreibungen anderer Personen abzurufen und zu lesen. Auf diese Weise kann der Teilnehmer auch zu Freunden der Person navigieren, auf die in der Beschreibung verwiesen wird.

3.3.6 Freunde hinzufügen

/BA6/

Ein wichtiger Bestandteil eines Online Social Networks ist es, Freundschafts- oder Bekanntschaftsbeziehungen abzubilden. Zu diesem Zweck soll es einer Person möglich sein, Freunde zu seiner Freundesliste in der persönlichen Beschreibung hinzuzufügen. Eine allgemeinere Variante davon, die auch auf andere Ressourcen anwendbar ist, ist das *liken* einer Internetseite.

3.3.7 Statusnachrichten veröffentlichen

/BA7/

Ein weit verbreitetes Kommunikationsmittel in Online Social Networks sind Statusnachrichten. Sie bilden einen Microblog über den eine Person Kurznachrichten für ihre Leser veröffentlichen kann. Auf diese Weise kann man (öffentlich) mit Freunden kommunizieren. Zusätzlich soll auch eine private Kommunikation ermöglicht werden, wobei Nachrichten nur einem geschlossenen Kreis zur Verfügung gestellt werden.

3.3.8 Medien-Artefakte veröffentlichen und Bilder taggen

/BA8/

Ebenso wie Statusnachrichten soll es auch möglich sein einfache Medieninhalte zu veröffentlichen. Dies könnten Bild-, Ton- oder Filmdateien sein. Es soll möglich sein, Bilder mit den persönlichen Beschreibungen darauf abgebildeter Personen zu verbinden (*taggen*).

3.3.9 Beliebige Internetseiten und Ressourcen teilen

/BA9/

Um die Möglichkeiten des Teilens nicht nur auf Statusnachrichten und Medien-Artefakte zu beschränken, soll es möglich sein seine Freunde auf beliebige Internetseiten hinzuweisen. Wenn eine Internetseite bereits RDF-Annotationen besitzt, sollen diese berücksichtigt werden. Auf diese Weise können Ressourcen beliebiger Dienste integriert werden und über diese diskutiert werden.

3.3.10 Ressourcen abonnieren und Abonnementverwaltung

/BA10/

Eine Person soll bestimmte Ressourcen (z. B. persönliche Beschreibungen, Statusnachrichten und Medien-Artefakte), die sie interessieren, abonnieren können. Freunde, die sie zu ihrer Freundesliste hinzufügt, abonniert die Person automatisch, ebenso wie Ressourcen die sie kommentiert. Der Person soll es auch möglich sein diese Abonnements für Ressourcen zu verwalten, das bedeutet neue Abonnements hinzufügen und bestehende entfernen zu können.

3.3.11 Aktualisierungen zu abonnierten Ressourcen empfangen

/BA11/

Eine Person soll stets Aktualisierungen zu den Ressourcen empfangen, die sie abonniert hat. Diese beinhalten neue Statusnachrichten und Medien-Artefakte der Freunde.

3.3.12 Ressourcen kommentieren

/BA12/

Es soll Personen möglich sein auf beliebige Ressourcen (dies können z. B. Statusnachrichten oder Medien-Artefakte sein) mit einem Kommentar zu antworten. Diesen Kommentar empfangen jeweils die Personen, welche die kommentierten Ressourcen abonniert haben.

3.3.13 Benachrichtigungen

/BA13/

Wenn eine Person zu der Freundesliste einer anderen Person hinzugefügt oder auf einem Bild getaggt wird, soll sie darüber benachrichtigt werden. Zu diesem Zweck wird ein Benachrichtigungssystem benötigt, das der Person Informationen über sie persönlich betreffende Aktionen gibt.

3.4 Zusammenfassung der Anforderungen

Um einen Überblick über die Notwendigkeit der einzelnen Benutzeranforderungen zu erlangen wird in Tab. 3.1 eine Übersicht darüber gegeben, welche Anforderungen zur Umsetzung welches Anwendungsfalls benötigt werden. Einige Anforderungen werden für alle Anwendungsfälle benötigt oder werden implizit vorausgesetzt. Solche Anforderungen werden in der Tabelle in Klammern dargestellt. Die letzte Zeile zeigt die Anzahl der Anwendungsfälle an, die eine bestimmte Benutzeranforderung begründen. Dies kann man als intuitives Maß für die Notwendigkeit einer Benutzeranforderung verwenden.

Privatsphäre ist ein Grundrecht, daher wird die Anforderung „Privatsphäre, Datenschutz und Redefreiheit“ in BA 1 auf S. 20 für alle Anwendungsfälle benötigt. Um ein Netzwerk zur Kommunikation zu nutzen, sind Erreichbarkeit und Stabilität notwendig. Daher wird auch die Anforderung „Zuverlässigkeit“ in BA 2 auf S. 21 für alle Anwendungsfälle benötigt. Jede an einem Online Social Network teilnehmende Person benötigt eine digitale Identität, welche durch eine persönliche Beschreibung zur Verfügung gestellt wird. Dies ist in der Anforderung „Persönliche Beschreibung anlegen und bearbeiten“ in BA 4 auf S. 22 formuliert.

	Privatsphäre, Datenschutz und Redefreiheit	Zuverlässigkeit	Kommunikation über Server- bzw. Anbietergrenzen hinweg	Persönliche Beschreibung anlegen und bearbeiten	Beschreibungen anderer Personen abrufen	Freunde hinzufügen	Statusnachrichten veröffentlichen	Medien-Artefakte veröffentlichen und Bilder taggen	Beliebige Internetseiten und Ressourcen teilen	Ressourcen abonnieren und Abonnementverwaltung	Aktualisierungen zu abonnierten Ressourcen empfangen	Ressourcen kommentieren	Benachrichtigungen
/AW1/	(×)	(×)	×	×				×		×	×	×	×
/AW2/	(×)	(×)		×	×	×	×	×		×	×		×
/AW3/	×	×	×	(×)			×	×	×			×	
/AW4/	(×)	(×)	×	×	×	×	×		×	×	×	×	×
Σ	4	4	3	4	2	2	3	3	2	3	3	3	3

Tabelle 3.1: Anforderungsmatrix: Übersicht der Anwendungsfälle und der zur Umsetzung benötigten Anforderungen

Anwendungsfall 1: „Foto teilen, taggen und kommentieren“ (S. 15)

Anwendungsfall 2: „Soziale Kommunikation und Freundschaft schließen“ (S. 16)

Anwendungsfall 3: „Investigativer Journalismus“ (S. 18)

Anwendungsfall 4: „Event organisieren“ (S. 19)

4 Bestandsaufnahme

In Sozialen Netzwerken findet heute ein großer Teil der Web-Aktivitäten statt. Insbesondere zentrale Ansätze haben dabei einen großen Zulauf (siehe Einführung Kap. 1 auf S. 7). Wie Yeung et al. (2009) in „Decentralization: The Future of Online Social Networking“ beschreiben, stellt uns dies vor einige Probleme. Einerseits sind diese Netze samt den enthaltenen Informationen von anderen Netzen abgegrenzt, andererseits lassen sie für ihre Teilnehmer kaum Kontrolle über die Nutzung der Informationen zu. Um den genannten Problemen zu entgegnen, ist, parallel zur steigenden Bedeutung von zentralen Online Social Networks, auch das Interesse an Ansätzen zum Aufbau verteilter Online Social Networks entstanden und immer größer geworden. Ansätze zum Aufbau verteilter Online Social Networks lassen sich grob in zwei Gruppen aufteilen, solche die Web 2.0-Techniken nutzen und solche die auf Semantic Web-Techniken beruhen (Tramp et al., 2012). Zentrale Ansätze werden an dieser Stelle nicht weiter betrachtet, da sie die Anforderungen „Privatsphäre, Datenschutz und Redefreiheit“ (BA 1 auf S. 20) und „Zuverlässigkeit“ (BA 2 auf S. 21) nicht erfüllen können und die Anforderung „Kommunikation über Server- bzw. Anbietergrenzen hinweg“ (BA 3 auf S. 21) momentan von keinem der in der Einführung (Kap. 1 auf S. 7) genannten Netzwerke erfüllt wird.

4.1 Web 2.0 Ansätze

Ein Ansatz, der durch eine erfolgreiche Spendenkampagne auf Kickstarter große Bekanntheit erlangt hat, ist das Diaspora-Projekt[1]. Das Diaspora-Netzwerk ist dabei aus einzelnen Knoten, den sogenannten *pods*, aufgebaut. Auf einem *pod* können mehrere Benutzer Profile anlegen und mit anderen Nutzern auf unterschiedlichen *pods* kommunizieren. Unterschiedliche *pods* kommunizieren untereinander mit einem proprietären „Federation Protocol“[2], die persönlichen Beschreibungen können mit dem WebFinger-Protokoll[3] von anderen *pods* abgerufen werden[4].

[1] Diaspora-Project: `http://diasporaproject.org/`, Diaspora auf Kickstarter: `http://www.kickstarter.com/projects/196017994/diaspora-the-personally-controlled-do-it-all-distr/`

[2] Beschreibung des *federation protocols* von Diaspora: `http://wiki.diaspora-project.org/wiki/Federation_Protocol_Overview/`

[3] WebFinger-Protokoll: `http://code.google.com/p/webfinger/wiki/WebFingerProtocol/`

[4] Blogeintrag zu Diaspora: `http://www.sarahmei.com/blog/2011/09/17/how-diaspora-connects-users/`

Ein weiterer Ansatz für ein verteiltes Online Social Network ist StatusNet[5]. StatusNet bietet einen Kurznachrichtendienst ähnlich wie Twitter an, bei dem die Statusnachrichten zwischen den verteilten Knoten über das OStatus-Protokoll[6] ausgetauscht werden. Wie auch bei Diaspora, wird WebFinger zur Abfrage der persönlichen Beschreibungen eingesetzt. Aktivitäten werden in Activity-Streams (vgl. Abschn. 2.4 auf S. 12) zusammengefasst und per PubSubHubbub verteilt[7]. Die größte Installation der StatusNet-Software ist identi.ca[8]. Am 1. Mai 2013 wurde eine Software-Umstellung für identi.ca von StatusNet auf pump.io[9] angekündigt.

Pump.io ist wie StatusNet ein verteiltes System. Es bietet aber im Gegensatz zu StatusNet nicht nur Statusnachrichten an, sondern erweitert das Angebot um weitere in Online Social Networks übliche Funktionen, wie z. B. das Teilen von Medien-Artefakten. Activity-Streams spielen bei pump.io eine wichtige Rolle und werden bei vielen Aktionen automatisch erzeugt. Neben Activity-Streams zum Austausch von Aktivitäten zwischen den Knoten wird auch WebFinger im Zusammenspiel mit OAuth zur Identifikation und Authentifizierung von Nutzern verwendet.[10] Abbildung 4.1 auf S. 28 zeigt eine Profilansicht auf dem pump.io-Knoten `http://pumprock.net`.

An der Universität Paderborn wurde LifeSocial.KOM, ein „P2P Framework for Social Networks“[11] (Graffi et al., 2011) entwickelt. Es stellt einen Rahmen zum Aufbau dezentraler Online Social Networks zur Verfügung und besteht aus einem *structured peer-to-peer overlay* über dem Internet, das zur verteilten Datenspeicherung verwendet wird. Im Gegensatz zu den bisher genannten Online Social Networks werden bei LifeSocial.KOM nicht alle Knoten in einem großen Netzwerk zusammengeschlossen, sondern es bildet immer eine Gruppe von Knoten ein abgegrenztes Netzwerk für eine bestimmte Nutzergruppe. Unterschiedliche für Online Social Networks typische Anwendungen können über Plug-Ins hinzugefügt werden.

4.2 Semantic Web Ansätze

Einen dezentralen und komplett über das Internet verteilten Ansatz für ein Online Social Network verfolgen verschiedene Semantic Web-Projekte. Die Hauptidee ist dabei nicht nur ein Userinterface mit Informationen zu den Personen zu veröffentlichen,

[5] StatusNet: `http://status.net/`

[6] OStatus-Protokoll: `http://www.w3.org/community/ostatus/wiki/Main_Page/`

[7] Interview mit StatusNet und pump.io Entwickler E. Prodromou: `http://ostatic.com/blog/evan-prodromou-speaks-on-the-future-of-statusnet/`

[8] identi.ca: `http://identi.ca/`, `http://status.net/wiki/ListOfServers#Main_server_installations`

[9] Ankündigung der Serverumstellung auf pump.io: `http://identi.ca/notice/100806687`, `http://identi.ca/doc/pumpio`, `http://pump.io/`

[10] „StatusNet, Identi.ca, and transitioning to pump.io“: `https://lwn.net/Articles/544347/`

[11] P2P Framework LifeSocial.KOM: `http://p2pframework.com/`

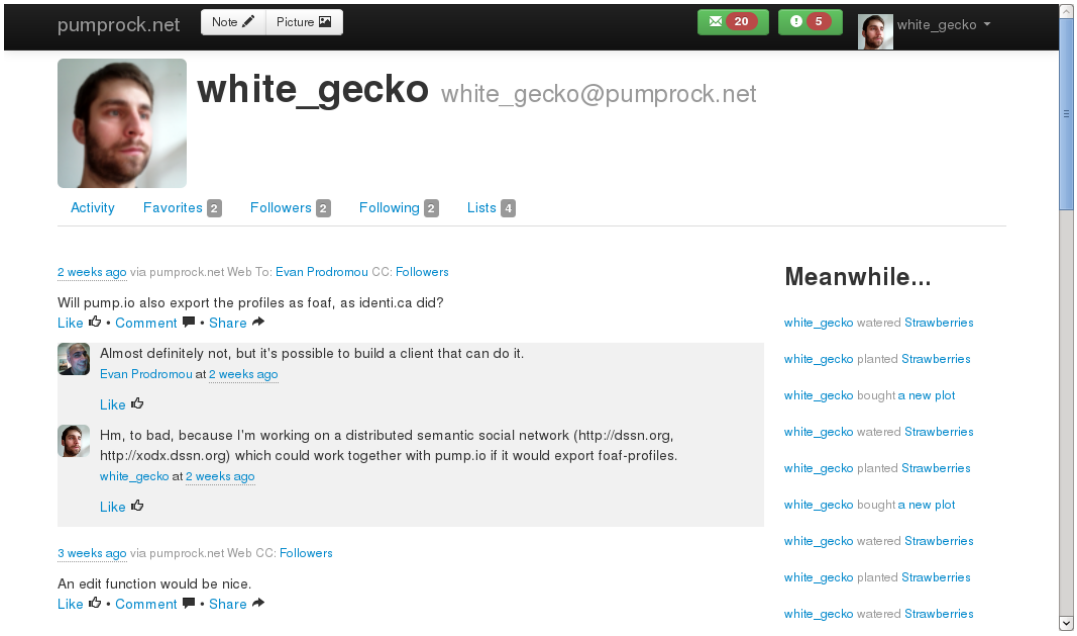


Abbildung 4.1: Die Profilansicht von Pump.io

sondern die zugrundeliegenden Daten strukturiert und maschinenlesbar im RDF zu repräsentieren. Die Ressourcen werden per Linked Data (Berners-Lee, 2009) abrufbar verteilt im Internet abgelegt, sodass jeder Teilnehmer seine Daten auf einem von ihm kontrollierten Server oder zumindest einem Server seines Vertrauens speichern kann. Weit verbreitete Vokabulare zur Repräsentation von Social Network-Daten sind `foaf` und `sioc` (Brickley & Miller, 2004; Breslin et al., 2005), sie stellen Klassen und Relationen zur Beschreibung von Personen, Beziehungen zwischen Personen und Online-Communities zur Verfügung. Das Online Social Network im Semantic Web reicht von der bloßen Repräsentation der persönlichen Beschreibungen der Teilnehmer und deren Beziehungen bis zur semantischen Auswertung der Daten und der Kommunikation zwischen den Teilnehmern.

Eine existierende Implementierung eines Dienstes zum Erstellen und Verwalten einer persönlichen Beschreibung und zum Bearbeiten der Freundesliste ist MyProfile[12]. Abbildung 4.2 auf S. 29 zeigt eine Profilansicht auf dem MyProfile-Server. Zusätzlich zur Verwaltung einer persönlichen Beschreibung bietet es für jeden Teilnehmer eine *wall* an. Auf diese können sowohl der Benutzer als auch seine Freunde lesend und schreibend zugreifen. Die *wall* ist ähnlich der Statusnachrichten-Funktion (BA 7 auf S. 22), bietet aber keine Kommunikation über Servergrenzen hinaus an (BA 3 auf S. 21). Über ein angepasstes Pingback-Protokoll wird eine Nachrichtenfunktion angeboten, die auch über Servergrenzen hinaus funktioniert. Aktivitäten werden in einer

[12] MyProfile Projektseite: `http://myprofile-project.org/`,
MyProfile Demo Instanz: `https://my-profile.eu/`

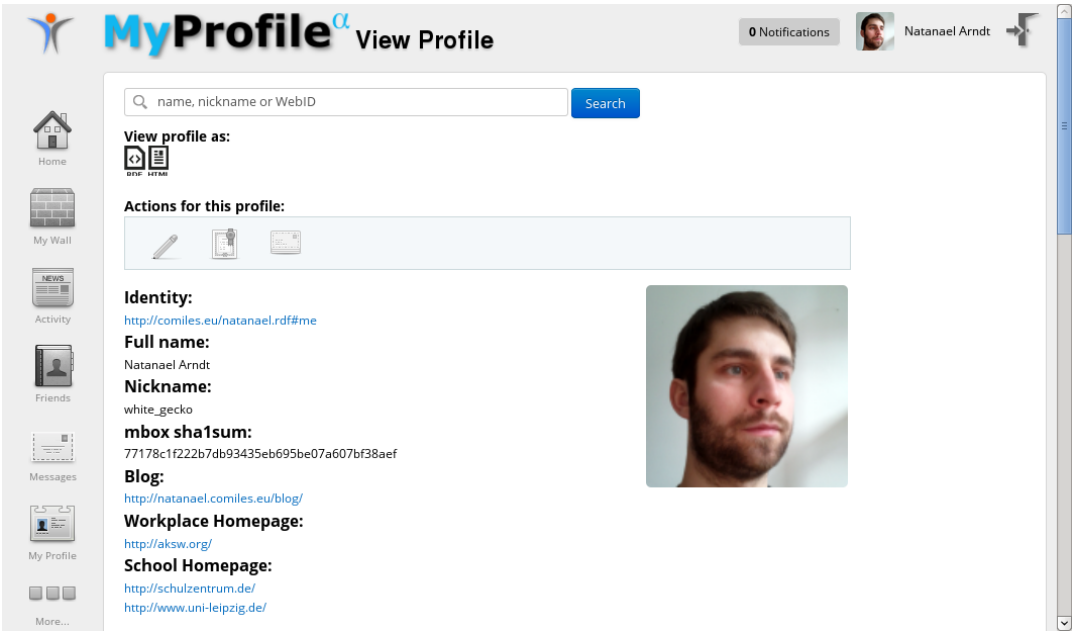


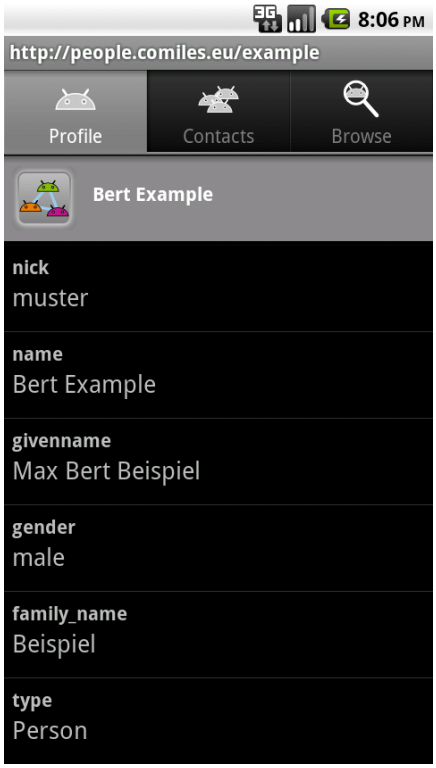
Abbildung 4.2: Die Profilansicht von MyProfile

Activity-Ansicht und einem Atom-Feed veröffentlicht, dieser ist allerdings ebenfalls nicht für eine verteilte Kommunikation verwendbar.

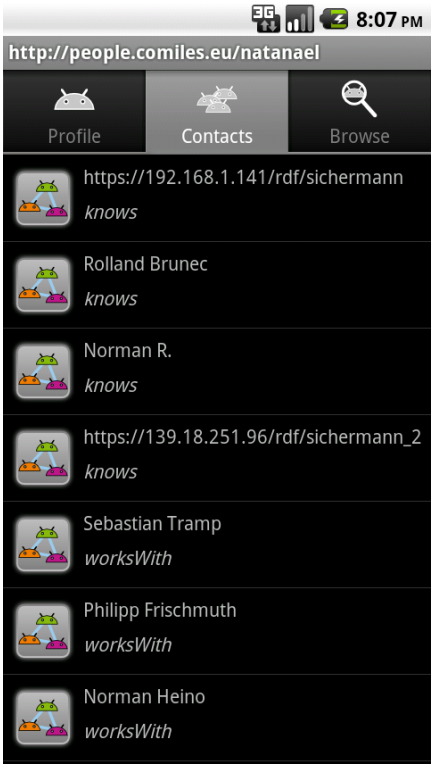
Um die Daten des Semantic Social Networks lesen zu können gibt es weiterhin Browser-Implementierungen. Für Android Systeme wurde der „FOAF/WebID Provider & Browser“ entwickelt (Arndt, 2010a,b; Tramp et al., 2011). Er bietet die Möglichkeit persönliche Beschreibungen anzuzeigen und durch die Freundesliste der Personen zu navigieren. Über eine Suchfunktion können weitere Personen gefunden, angezeigt und zu der eigenen Freundesliste hinzugefügt werden. Abbildung 4.3a auf S. 30 zeigt die Profilansicht und Abbildung 4.3b die Freundeslisten-Ansicht des „FOAF/WebID Provider & Browser“ auf Android. Darüber hinaus ist es möglich eine persönliche Beschreibung zu initialisieren und die darin referenzierten Freunde in das Android-Systemadressbuch zu importieren.

Bei den bisher genannten Ansätzen sind aber noch einige Details zum Zusammenspiel der einzelnen Teile des Semantic Social Networks nicht geklärt. Eine komplette Architektur mit Daten-, Protokoll-, Anwendungs- und Service-Ebene wird in „An Architecture of a Distributed Semantic Social Network“ (Tramp et al., 2012) beschrieben. Das Distributed Semantic Social Network (DSSN) setzt dabei zusätzlich zu anderen Semantic Web-Projekten auf bereits im Web 2.0 etablierte Techniken, wie (Semantic) Pingback (Tramp et al., 2010) und Activity-Streams (vgl. Abschn. 2.4 auf S. 12), die mithilfe des Publish-Subscribe-Protokolls PubSubHubbub[13] (vgl. Abschn. 2.3 auf

[13] PubSubHubbub: `http://pubsubhubbub.appspot.com/`



(a) Eine Profilansicht mit dem „FOAF/WebID Provider & Browser“ auf Android



(b) Eine Freundesliste mit dem „FOAF/WebID Provider & Browser“ auf Android

Abbildung 4.3: Ansichten der DSSN-Browseranwendung „FOAF/WebID Provider & Browser“ für Android

S. 11) verteilt werden. In der Arbeit „Konzept und Implementierung aktivitätsbasierter Kommunikation für verteilte semantische Soziale Netzwerke“ (Radtke, 2013) wurde die Implementierung der *activities* und Activity-Streams näher untersucht und durchgeführt.

Die Ansätze auf Basis von Semantic Web-Technologien haben ein größeres Potential, da sie flexibler sind und eine bessere Erweiterbarkeit ermöglichen. Aufgrund des direkten Zugriffs auf die Daten und der maschinenlesbaren semantischen Auszeichnung der Daten können leicht andere Services und Anwendungsfälle umgesetzt und integriert werden. Die Integration der einzelnen Services ist dabei auf Daten-Ebene im gesamten Internet möglich und nicht auf die Domäne der sozialen Vernetzung und Kommunikation beschränkt. In dieser Arbeit wird daher auf die bestehende Architektur aus „An Architecture of a Distributed Semantic Social Network“ (Tramp et al., 2012) aufgebaut und ein Knoten implementiert, der die darin beschriebenen Techniken umsetzt.

5 Spezifikation

> „System requirements are more detailed descriptions of the software system's functions, services, and operational constraints. The system requirements document (sometimes called a functional specification) should define exactly what is to be implemented.“ (Sommerville, 2011, S. 83)

In den folgenden Abschnitten werden die Funktionen und Schnittstellen des entstehenden Online Social Networks entsprechend den Benutzeranforderungen aus Kap. 3 auf S. 14 spezifiziert. Die dabei entstehende Software zum Betrieb eines Knotens heißt Xodx. Da es sich um ein verteiltes System handelt, ist im DSSN die Kommunikation der Knoten untereinander besonders wichtig. Ebenfalls wird eine Benutzerschnittstelle als Browseranwendung zur Kommunikation im DSSN beschrieben. Zuletzt werden die Anforderungen der Software an ihre Ausführungsumgebung in Abschn. 5.6 auf S. 46 spezifiziert.

5.1 Terminologie

Um ein einheitliches Verständnis dieser Arbeit zu ermöglichen werden hier, zusätzlich zu den in Abschn. 3.1 auf S. 14 definierten Rollen, häufig verwendete Begriffe definiert bzw. erweitert. Gelegentlich wird auf bestehende Definitionen von RDF-Typen verwiesen um eine semantische Einordnung der Begriffe zu vereinfachen. Die dabei verwendeten Namespace-Abkürzungen sind im Turtle-Format in Lst. 5.1 auf S. 32 zusammengefasst. Das `dssn`-Vokabular wurde im Rahmen dieser Arbeit um für die Umsetzung notwendige Klassen und Relationen erweitert.

Person ist eine Einheit zur Beschreibung einer natürlichen Person (vgl. `foaf:Person`). Dabei wird einer Person eindeutig ein URI zugewiesen, dieser URI wird auch WebID genannt. Der Begriff WebID soll in dieser Arbeit aber vermieden werden, da er zur Vermischung der Aspekte Identifikation (WebID), Authentifizierung (WebID/FOAF+SSL) und Autorisierung (ACL)[1] führen könnte.

[1] Mailingliste `public-webid@w3.org`: Melvin Carvalho: „WebID Simple“: `http://lists.w3.org/Archives/Public/public-webid/2013Apr/0048.html`

Persönliche Beschreibung ist die Zusammenfassung der Aussagen zur Beschreibung einer `foaf:Person`. Sie drückt Eigenschaften und Beziehungen der Person aus. Die persönliche Beschreibung einer Person wird (u. U. zusammen mit anderen Beschreibungen) in einem Profildokument mit einer Ressource vom Typ `foaf:PersonalProfileDocument` ausgeliefert.

Benutzerkonto ist eine Organisationseinheit auf einem Knoten der den Zugang eines Agenten bzw. einer Person zu dem Dienst sicherstellt und regelt. Ein Agent authentifiziert sich einem Knoten gegenüber als Inhaber eines Benutzerkontos (*account*) z. B. unter Verwendung eines Benutzernamens und Passworts. (vgl. `sioc:UserAccount`)

Medien-Artefakt ist eine Bild-, Ton- oder Filmdatei und Beschreibung des Medieninhalts. Medien-Artefakte können von einer Person mit anderen Personen geteilt werden.

Online Social Network ist eine zentrale oder dezentrale Anwendung im Internet zur Unterstützung sozialer Kommunikation. In der Regel umfasst dies eine Abbildung der Freundschaftsbeziehungen zwischen den Personen und verschiedene Anwendungen zur Interaktion der teilnehmenden Personen.

DSSN Das Distributed Semantic Social Network (DSSN) ist ein Online Social Network. Es ist verteilt organisiert und verwendet Semantic Web-Technologien zur Repräsenation der Daten sowie verschiedene Protokolle zur Kommunikation der Knoten untereinander.

```
1  # Allgemeine Namespace-Definitionen
2  @prefix aair: <http://xmlns.notu.be/aair#> .
3  @prefix activity: <http://activitystrea.ms/spec/1.0/> .
4  @prefix atom: <http://www.w3.org/2005/Atom/> .
5  @prefix dct: <http://purl.org/dc/terms/> .
6  @prefix dssn: <http://purl.org/net/dssn/> .
7  @prefix foaf: <http://xmlns.com/foaf/0.1/> .
8  @prefix ov: <http://open.vocab.org/docs/> .
9  @prefix owl: <http://www.w3.org/2002/07/owl#> .
10 @prefix pingback: <http://purl.org/net/pingback/> .
11 @prefix rdf: <http://www.w3.org/1999/02/22-rdf-syntax-ns#> .
12 @prefix rdfs: <http://www.w3.org/2000/01/rdf-schema#> .
13 @prefix rel: <http://purl.org/vocab/relationship/> .
14 @prefix sioc: <http://rdfs.org/sioc/ns#> .
15 @prefix sioct: <http://rdfs.org/sioc/types#> .
16 @prefix xsd: <http://www.w3.org/2001/XMLSchema#> .
17
18
```

```
19 # Namespaces der Xodx-Beispiel-Instanzen
20 @prefix fb: <http://fb.comiles.eu/> .
21 @prefix xo: <http://example.org/xodx/> .
```

Listing 5.1: Definition der verwendeten Namespace-Abkürzungen

5.2 Aufbau der entstehenden Software

Der hier entworfene DSSN Knoten besitzt eine Benutzerschnittstelle zur Darstellung im Webbrowser und einen Unterbau, der die Datenspeicherung und Kommunikation im Netzwerk bereitstellt. Eine wichtige Anforderung für das Netzwerk ist die Kommunikation über Server- bzw. Anbietergrenzen hinweg (BA 3 auf S. 21). Diese Server- und Anbieterunabhängigkeit soll durch die direkte Bereitstellung der zugrunde liegenden Daten per Linked Data (siehe Abschn. 2.1 auf S. 10) und eine Entkopplung der einzelnen Dienste im Netzwerk (*service decoupling*) erreicht werden. Auf der anderen Seite ist eine Kommunikation der Server untereinander nötig. Um diese möglichst einfach sicher zu stellen, werden wenige, bereits weit verbreitete Protokolle eingesetzt. Durch die Beschränkung auf wenige etablierte Protokolle wird zusätzlich eine einfache Erweiterbarkeit des Netzes ermöglicht. Das gesamte Netz wird zur Umsetzung dieser Prinzipien in mehrere Ebenen aufgeteilt. Diese Ebenen-Architektur spiegelt sich auch in den Knoten wieder.

5.2.1 Ebenen-Architektur

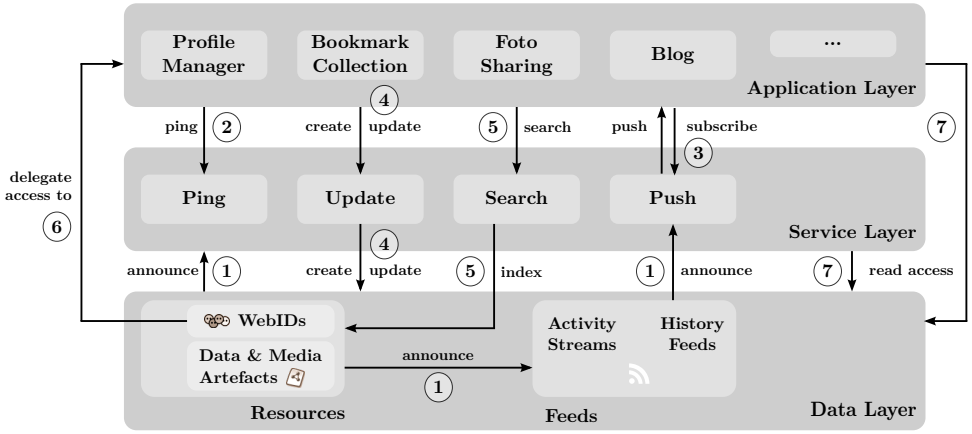


Abbildung 5.1: Aufbau und Kommunikation der Dienste des DSSN. Quelle: Tramp et al. (2012).

Das DSSN ist in mehrere Ebenen (*layer*) unterteilt: Daten-, Protokoll- , Anwendungs- und Service-Ebene. Abbildung 5.1 zeigt die Interaktionen zwischen den Ebenen mit einigen Beispielanwendungen und -diensten. Durch Pfeile mit Nummern wird das konkrete Zusammenspiel der einzelnen Teile innerhalb und zwischen den Ebenen dargestellt. (1) Ressourcen verweisen auf ihre Activity- und History-Feeds, Ressourcen und Feeds verweisen auf Services, über die sie angesprochen resp. publiziert werden. (2) Anwendungen senden Ping-Anfragen, um Ressourcen um sie betreffende Relationen zu erweitern, und so das Linked Data-Netzwerk aufzubauen und zu erweitern. (3) Anwendungen abonnieren Feeds und werden über Änderungen benachrichtigt. (4) Update-Services können Ressourcen und Feeds auf Anfrage von autorisierten Anwendungen aktualisieren. (5) Persönliche und globale Such-Dienste können Ressourcen indizieren und auf Anfragen von Anwendungen antworten. (6) Anwendungen können Zugriff auf Ressourcen und Dienste erhalten, um im Namen eines Teilnehmers zu handeln. (7) Die meisten Anfragen sind Standard-HTTP-Anfragen gemäß den Linked Data-Richtlinien. (Tramp et al., 2012)

Ein Knoten kann Dienste in einer oder mehreren Ebenen zur Verfügung stellen und ebenso kann es in jeder Ebene mehrere Knoten geben, die den gleichen Dienst anbieten. Die in dieser Arbeit entwickelte Knoten-Implementierung soll zusammenhängende Dienste in allen Ebenen zur Verfügung stellen. Trotzdem sollen die Dienste auch unabhängig voneinander angesprochen werden können. Der konkretere Aufbau des Knotens und die Kommunikation mit anderen Knoten ist in Abb. 5.2 auf S. 35 grafisch dargestellt. Die Interaktionen (und deren Nummerierung) sind dabei die gleichen wie in Abb. 5.1 auf S. 33 mit folgenden Änderungen: (3') Anwendungen publizieren Aktualisierungen des Activity-Streams über den PubSubHubbub-Hub. (4') Der Einsatz eines Update-Services entfällt, da die Anwendungen innerhalb des Xodx-Knotens selbst die Änderungen an den Ressourcen bzw. Feeds vornehmen.

Durch die verteilte Architektur und die unabhängige Arbeitsweise der einzelnen Dienste wird sowohl ein zentraler Angriffspunkt (*single point of attack*) als auch ein einzelner Fehlerpunkt (*single point of failure*) vermieden. Dies führt zu einer höheren Zuverlässigkeit des gesamten Netzwerks (Anforderung „Zuverlässigkeit“ in BA 2 auf S. 21). Zusätzlich ist auf diese Weise jeder Teilnehmer in der Position entscheiden zu können, auf welchem Daten-Knoten er seine Daten hinterlegt, welchen Anwendungs-Knoten er Zugriff gewährt und welche Protokoll-Knoten und Service-Knoten er nutzt. Diese können dabei auf einem eigenen selbstbetriebenen Server oder bei einem anderen vertrauenswürdigen Anbieter (z. B. einem Freund, Verein oder dem Arbeitgeber) installiert sein. Durch die Vermeidung eines *single point of attack* und die Auswahlmöglichkeiten des Teilnehmers können so Privatsphäre, Datenschutz und Redefreiheit (Anforderung „Privatsphäre, Datenschutz und Redefreiheit“ in BA 1 auf S. 20) ermöglicht werden.

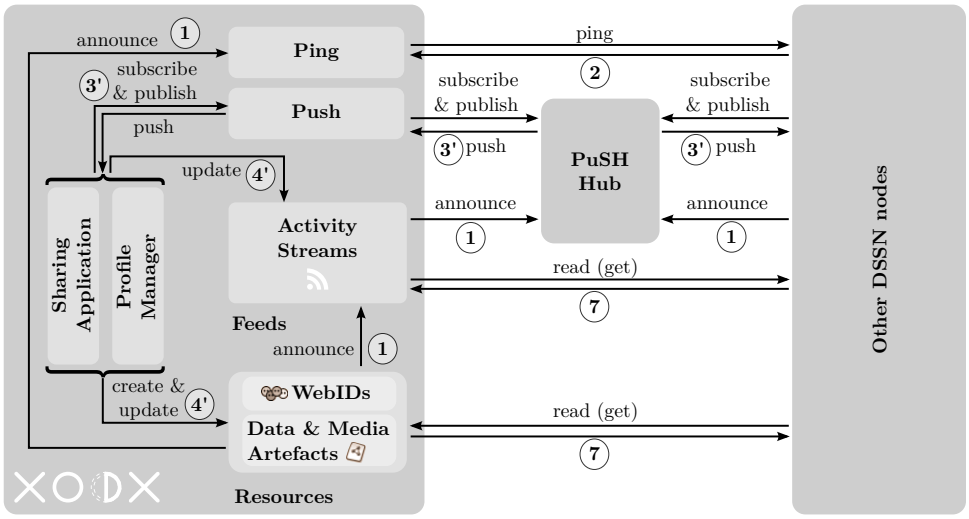


Abbildung 5.2: Aufbau und Kommunikation eines Xodx-Knotens mit anderen Knoten im DSSN. Mit Bezug auf Abb. 5.1.

5.3 Verwendetes Datenmodell – Daten-Ebene

In der Daten-Ebene (*data layer*) werden die RDF-Ressourcen, Medien-Artefakte und die zugehörigen Activity-Feeds gespeichert. Eine Person sucht sich dazu einen Daten-Knoten aus, auf dem ihre persönliche Beschreibung, Freundesliste, Statusnachrichten und Medien-Artefakte hinterlegt werden. Alle Ressourcen werden über einen URI eindeutig referenziert und sind mit diesem per Linked Data abrufbar.

Im DSSN können beliebige Ressourcen kommentiert werden. Da dies für alle folgenden Beispiele gilt werden die dafür nötigen Relationen an dieser Stelle nur einmal genannt. Zu diesem Zweck wird jeder Ressource ein Activity-Stream bzw. -Feed zugeordnet. Dies geschieht mit der Relation `dssn:activityFeed`. Um eine Ressource über einen neuen Kommentar zu benachrichtigen muss für sie ein Ping-Service angegeben sein, dies geschieht mit der Relation `pingback:to`. Auf die Verwendung der Relationen wird in der Spezifikation der Protokoll-Ebene (Abschn. 5.4 auf S. 41) näher eingegangen.

5.3.1 Persönliche Beschreibung und Beziehung zwischen Personen

Die persönliche Beschreibung einer Person wird im `foaf`-Vokabular in Form einer Ressource vom Typ `foaf:Person` formuliert. Listing 5.2 auf S. 36 zeigt beispielhaft eine persönliche Beschreibung. In der ersten Zeile wird die Klasse der Ressource definiert,

danach folgt eine Eigenschaft `foaf:name`, die den Namen der Person angibt. In Zeile 3 wird die Person mit einem Benutzerkonto assoziiert. Bekanntschaftsbeziehungen zwischen Personen werden durch die Verwendung der Relation `foaf:knows` beschrieben (im Beispiel in Zeile 4). Es können auch beliebige andere Relationen verwendet werden. Zum Beispiel ist es auch möglich, statt der allgemeinen Relation `foaf:knows`, speziellere Angaben aus dem `rel`-Vokabular zu verwenden.

```
1 fb:?c=person&id=natanael rdf:type foaf:Person ;
2                          foaf:name "Natanael Arndt" ;
3                          foaf:account fb:?c=user&id=natanael ;
4                          foaf:knows <http://sebastian.tramp.name>, xo:?c=person&id=↲
                             norman ;
5                          dssn:activityFeed fb:?c=feed&a=getFeed&uri=http%3A%2F%2Ffb.↲
                             comiles.eu%2Fxodx%2F%3Fc%3Dperson%26id%3Dnatanael ;
6                          pingback:to fb:?c=pingback&a=ping .
```

Listing 5.2: Beispiel einer persönlichen Beschreibung in Turtle

Bei der Anfrage einer `foaf:Person` per Linked Data wird der Anfragende durch *content negotiation* (vgl. Abschn. 5.4.1 auf S. 41) zu einem entsprechenden `foaf:PersonalProfileDocument` weitergeleitet, das als `foaf:primaryTopic` die angefragte `foaf:Person` beinhaltet. Damit entspricht die Auslieferung dem Entwurf der WebID-Spezifikation nach Sporny et al. (2011).

Das hier beschriebene Datenmodell dient zur Umsetzung der Anforderungen „Persönliche Beschreibung anlegen und bearbeiten“ in BA 4 auf S. 22, „Beschreibungen anderer Personen abrufen“ in BA 5 auf S. 22 und „Freunde hinzufügen“ in BA 6 auf S. 22.

5.3.2 Darstellung von Statusnachrichten, Medien-Artefakten und Kommentaren

Statusnachrichten werden, wie in Lst. 5.3 auf S. 37 beispielhaft dargestellt, im `sioc`-Vokabular als `sioc:Post`-Ressourcen formuliert. Da sie außerdem in Activity-Feeds eingetragen werden, haben sie zusätzlich die Klasse `aair:Note`. Der eigentliche Textinhalt der Statusnachricht wird mit der Eigenschaft `sioc:content` und zusätzlich mit der Eigenschaft `aair:content` angegeben. Die Eigenschaft `sioc:created_at` gibt das Erstellungsdatum der Statusnachricht an und über die Relation `foaf:maker` wird die Person des Autors verknüpft. Die Relationen `dssn:activityFeed` und `pingback:to` dienen wieder zur Vernetzung im DSSN.

Die doppelten Klassenzuordnungen der Ressourcen und die doppelte Angabe des Inhalts der Statusnachrichten, im `sioc`- resp. `foaf`- und `aair`-Vokabular wurden gewählt,

da einerseits das sioc- und das foaf-Vokabular größere Akzeptanz gefunden haben und andererseits die Einträge des Activity-Feeds im aair-Vokabular korrekt abgebildet werden sollen, um die Kompatibilität mit anderen Nutzern von Activity-Streams zu wahren.

```
1 fb:?c=resource&id=c018 rdf:type sioc:Post, aair:Note ;
2                        sioc:content "Ich sitze auf dem Balkon" ;
3                        aair:content "Ich sitze auf dem Balkon" ;
4                        sioc:created_at "2013-05-06T14:39:54+02:00"^^xsd:dateTime ;
5                        foaf:maker fb:?c=person&id=natanael ;
6                        dssn:activityFeed fb:?c=feed&a=getFeed&uri=http%3A%2F%2Ffb.↲
                           comiles.eu%2F%3Fc%3Dresource%26id%3Dc018 ;
7                        pingback:to fb:?c=pingback&a=ping .
```

Listing 5.3: Beispiel einer Statusnachricht in Turtle

Medien-Artefakte können verschiedene Typen haben. In Lst. 5.4 wird beispielhaft ein Artefakt vom Typ foaf:Image und aair:Photo gezeigt. Der MIME-Type wird in Zeile 2 mit der Eigenschaft ov:hasContentType angegeben. Die Relation aair:largerImage verweist auf die eigentliche Bilddatei. Durch Hinzufügen einer foaf:depicts-Relation können Personen oder Dinge in Bildern markiert werden. Die weiteren Eigenschaften und Relationen werden wie in den zuvor beschrieben Ressourcen verwendet.

```
1 fb:?c=resource&id=6d84 rdf:type foaf:Image, aair:Photo ;
2                        ov:hasContentType "image/jpeg" ;
3                        aair:largerImage fb:?c=resource&a=img&id=6d84 ;
4                        foaf:depicts fb:?c=person&id=natanael ;
5                        sioc:created_at "2013-04-26T16:24:03+02:00"^^xsd:dateTime ;
6                        foaf:maker fb:?c=person&id=natanael ;
7                        dssn:activityFeed fb:?c=feed&a=getFeed&uri=http%3A%2F%2Ffb.↲
                           comiles.eu%2F%3Fc%3Dresource%26id%3D6d84 ;
8                        pingback:to fb:?c=pingback&a=ping .
```

Listing 5.4: Beispiel der Beschreibung eines Medien-Artefakts (Bild) in Turtle

Kommentare zu beliebigen Ressourcen können als sioct:Comment bzw. aair:Comment erstellt werden (Lst. 5.5 auf S. 38). Die Eigenschaft aair:commenter verweist dabei auf den Kommentator. Da es eine owl:DatatypeProperty ist, also eine Eigenschaft die ein Literal erwartet, enthält sie den Namen des Kommentators. Ein Verweis auf die Person wird durch die Relation foaf:maker ausgedrückt. Die Relation sioc :reply_of verweist auf die kommentierte Ressource. Die Eigenschaften sioc:content und aair:content enthalten, wie bei einer Statusnachricht, den eigentlichen Text des Kommentars. Die weiteren Eigenschaften und Relationen werden wie zuvor beschrieben verwendet. Die kommentierte Ressource kann zusätzlich mit der Relation sioc :has_reply auf den Kommentar verweisen. Diese muss nicht auf dem selben Knoten wie der Kommentar gespeichert sein.

```
xo:?c=resource&id=43bc rdf:type sioct:Comment, aair:Comment ;
                       sioc:reply_of fb:?c=resource&id=c018 ;
                       sioc:content "Das ist ja toll" ;
                       aair:content "Das ist ja toll" ;
                       sioc:created_at "2013-05-09T23:03:20+02:00"^^xsd:dateTime ;
                       aair:commenter "Norman" ;
                       foaf:maker xo:?c=person&id=norman ;
                       dssn:activityFeed xo:?c=feed&a=getFeed&uri=http%3A%2F%2↲
                         Fexample.org%2Fxodx%2F%3Fc%3Dresource%26id%3D43bc ;
                       pingback:to xo:?c=pingback&a=ping .
```

Listing 5.5: Beispiel eines Kommentars in Turtle

Die hier beschriebenen Datenmodelle dienen der Umsetzung der Anforderungen „Statusnachrichten veröffentlichen“ in BA 7 auf S. 22, „Medien-Artefakte veröffentlichen und Bilder taggen“ in BA 8 auf S. 22 und „Ressourcen kommentieren“ in BA 12 auf S. 23.

5.3.3 Aufbau des Activity-Streams und -Feeds

Zum Erfassen der Handlungen einer Person wird ein Activity-Stream aufgebaut. Er setzt sich aus mehreren Aktivitäten zusammen, wobei eine Aktivität eine Handlung darstellt und aus *actor* (Handelnder), *verb* (Tätigkeit) und *object* (Objekt der Handlung) besteht. Dies sind unter anderem „Eine Person fügt eine andere Person zu ihrer Freundesliste hinzu“, „Eine Person verfasst eine Statusnachricht“, „Eine Person teilt einen Link (URL)“ oder „Eine Person teilt ein Medien-Artefakt“.

Der Aufbau eines Activity-Streams ist im „Activity Base Schema“ (Snell et al., 2012), „Json Activity Streams 1.0“ (Snell et al., 2011) und „Atom Activity Streams 1.0“ (Atkins et al., 2011) spezifiziert. Um den Activity-Stream im RDF-Datenmodell abzubilden wird das „Atom Activity Streams RDF Mapping“ (Minno & Palmisano, 2010) verwendet. Eine Aktivität kann als Ressource vom Typ `aair:Activity` abgebildet werden, siehe Listing 5.6. Der Zeitpunkt, an dem die Activity stattgefunden hat, wird durch die Eigenschaft `atom:published` angegeben. Eine Verknüpfung zu dem *actor* (im Beispiel die Person `fb:?c=person&id=natanael`) kann mit der Relation `aair:activityActor` hergestellt werden. Als Tätigkeit wird hier in Zeile 4 das allgemeine Verb `aair:Post` angegeben. Das Objekt im Beispiel ist die Ressource `fb:?c=resource&id=c018` aus Lst. 5.3 auf S. 37.

```
fb:?c=resource&id=4af8 rdf:type aair:Activity ;
                       atom:published "2013-05-06T14:39:54+02:00"^^xsd:dateTime ;
                       aair:activityActor fb:?c=person&id=natanael ;
                       aair:activityVerb aair:Post ;
```

```
5                       aair:activityObject fb:?c=resource&id=c018 ;
6                       dssn:activityFeed fb:?c=feed&a=getFeed&uri=http%3A%2F%2Ffb.↲
                          comiles.eu%2F%3Fc%3Dresource%26id%3D4af8 ;
7                       pingback:to fb:?c=pingback&a=ping .
```

Listing 5.6: Beispiel einer *activity* in Turtle

Die Publikation des Activity-Streams findet außer in RDF auch in Form eines Atom-Feeds statt. Listing 5.7 zeigt die Aktivität aus Lst. 5.6, wie sie in einem Atom-Feed als `entry` dargestellt wird. Zur Erstellung des `entry` wird ein beschreibender Titel (Zeile 7) und Inhalt (Zeile 23) generiert. Als `id` wird der URI der Aktivitäts-Ressource verwendet (Zeile 8), der auch in Zeile 9 verlinkt ist. Die Datumsangaben für `published` und `updated` (Zeile 10f) sind der Wert der Eigenschaft `atom:published`. In dem Feld `author` (Zeilen 12–15) werden der Name und URI der erstellenden Person angegeben. Das `activity:verb` in Zeile 16 wird aus der Relation `aair:activityVerb` gelesen. Dies können beliebige URIs zur Beschreibung der Tätigkeit sein. Stammt das Verb aus dem `aair`-Namespace, kann es durch Angabe des letzten Teils des URI abgekürzt werden. Das `activity:object` (Zeilen 17–22) ist das Objekt der Relation `aair:activityObject`. Die `id` ist wieder der URI der Objekt-Ressource und `published` gibt das Erstellungsdatum aus der Eigenschaft `sioc:created_at` an. Das Objekt kann von unterschiedlichen RDF-Typen sein, im Beispiel aus Lst. 5.3 auf S. 37 hat es die Typen `sioc:Post` und `aair:Note`. Ist ein Typ aus dem `aair`-Namespace angegeben so wird dieser als `activity:object-type` angegeben (er kann wie das Verb abgekürzt werden), andernfalls wird ein beliebiger angegebener Typ verwendet. Hat das Objekt eine Eigenschaft `aair:content`, wird zusätzlich ein `content`-Eintrag mit dem entsprechenden Wert angegeben.

```
1  <?xml version="1.0" encoding="utf-8"?>
2  <feed xmlns="http://www.w3.org/2005/Atom" xmlns:activity="http://activitystrea.ms/↲
     schema/1.0/">
3     <title>Activity Feed of Natanael Arndt</title>
4     <link rel="hub" href="http://pubsubhubbub.appspot.com"/>
5     ...
6     <entry>
7         <title>&quot;Natanael Arndt posted http://xmlns.notu.be/aair#Note&quot;</↲
            title>
8         <id>http://fb.comiles.eu/?c=resource&amp;id=4af8</id>
9         <link href="http://fb.comiles.eu/?c=resource&amp;id=4af8" />
10        <published>2013-05-06T14:39:54+02:00</published>
11        <updated>2013-05-06T14:39:54+02:00</updated>
12        <author>
13            <name>Natanael Arndt</name>
14            <uri>http://fb.comiles.eu/?c=person&amp;id=natanael</uri>
15        </author>
16        <activity:verb>post</activity:verb>
```

```
17          <activity:object>
18              <id>http://fb.comiles.eu/?c=resource&amp;id=c018</id>
19              <published>2013-05-06T14:39:54+02:00</published>
20              <activity:object-type>note</activity:object-type>
21              <content>Ich sitze auf dem Balkon</content>
22          </activity:object>
23          <content>New Activity: &quot;Natanael Arndt posted Note&quot;: Ich sitze auf↲
              dem Balkon</content>
24      </entry>
25  </feed>
```

Listing 5.7: Darstellung der *activity* aus Lst. 5.6 als *entry* in einem Atom-Feed

Die hier beschriebenen Datenmodelle dienen zur Umsetzung der Anforderungen „Statusnachrichten veröffentlichen“ in BA 7 auf S. 22, „Medien-Artefakte veröffentlichen und Bilder taggen“ in BA 8 auf S. 22, „Ressourcen abonnieren und Abonnementverwaltung“ in BA 10 auf S. 23, „Aktualisierungen zu abonnierten Ressourcen empfangen“ in BA 11 auf S. 23 und „Ressourcen kommentieren“ in BA 12 auf S. 23.

5.3.4 Darstellung eines Abonnements

Wenn Personen bzw. deren Benutzerkonten auf einem entfernten Knoten Ressourcen abonnieren, muss der eigene Knoten diese Abonnements verwalten können und eingehende Aktualisierungen der entsprechenden Person zuordnen (Lst. 5.8). Das Benutzerkonto der Person (Zeile 1f), die einen bestimmten Feed abonniert hat, verweist auf eine Ressource vom Typ `dssn:Subscription` (Zeile 3 bzw. 5). Das Modell des Abonnements (*subscription*) ist sehr stark an das PubSubHubbub-Protokoll (siehe Abschn. 2.3 auf S. 11) angelehnt. Der Feed, der abonniert wird, wird mit der Relation `dssn:subscriptionTopic` angegeben. Die Relation `dssn:subscriptionHub` verweist auf den verwendeten PubSubHubbub-Hub bzw. -Service, über den der Feed publiziert wird. Die Relation `dssn:subscriptionCallback` gibt die für PubSubHubbub notwendige Callback-Methode an, an die der Hub später Aktualisierungen sendet.

```
1  fb:?c=user&id=natanael a sioc:UserAccount ;
2                         sioc:account_of fb:?c=person&id=natanael ;
3                         dssn:subscribedTo fb:?c=resource&id=d8bf .
4
5  fb:?c=resource&id=d8bf rdf:type dssn:Subscription ;
6                         dssn:subscriptionTopic fb:?c=feed&a=getFeed&uri=http%3A%2F%2↲
                            Ffb.comiles.eu%2F%3Fc%3Dperson%26id%3Dnatanael ;
7                         dssn:subscriptionCallback fb:?c=push&a=callback ;
8                         dssn:subscriptionHub <http://pubsubhubbub.appspot.com> .
```

Listing 5.8: Darstellung eines Abonnements

Das hier beschriebene Datenmodell dient zur Umsetzung der Anforderungen „Ressourcen abonnieren und Abonnementverwaltung“ in BA 10 auf S. 23 und „Aktualisierungen zu abonnierten Ressourcen empfangen“ in BA 11 auf S. 23.

5.3.5 Darstellung einer Benachrichtigung

Um eine Person bzw. deren Benutzerkonto über sie betreffende Ereignisse zu benachrichtigen werden Ressourcen vom Typ `dssn:Notification` angelegt (Lst. 5.9). Das zu benachrichtigende Benutzerkonto wird über die Relation `dssn:notify` verknüpft. Die Eigenschaft `sioc:content` gibt einen beschreibenden Text an, der bei der Ausgabe der Benachrichtigung angezeigt wird. Durch die Relation `dct:references` kann eine Ressource angegeben werden, die mit der Benachrichtigung in Verbindung steht, z. B. eine Aktivität.

```
1 fb:?c=resource&id=h23b rdf:type dssn:Notification ;
2                        dssn:notify fb:?c=user&id=natanael ;
3                        sioc:content "You have been tagged on a Picture" ;
4                        dct:references fb:?c=resource&id=3bc2 .
```

Listing 5.9: Darstellung einer Benachrichtigung

Das hier beschriebene Datenmodell dient zur Umsetzung der Anforderungen „Ressourcen abonnieren und Abonnementverwaltung“ in BA 10 auf S. 23 und „Benachrichtigungen“ in BA 13 auf S. 23.

5.4 Eingesetzte Protokolle – Protokoll-Ebene

Die Protokoll-Ebene dient zur Kommunikation der einzelnen Dienste im DSSN untereinander. Dabei werden außer dem überall zugrunde liegenden *hypertext transfer protocoll* (HTTP) andere Protokolle mit unterschiedlichen Eigenschaften und Funktionen eingesetzt. Diese sind Linked Data, Semantic Pingback und PubSubHubbub.

5.4.1 Linked Data und Content Negotiation

Das Linked Data-Protokoll bzw. die Linked Data-Prinzipien dienen zur Abfrage der Ressourcen von einem Server. Dabei wird der URI der Ressource per HTTP angefragt. Als Antwort wird eine maschinenlesbare Repräsentation der Ressourcenbeschreibung erwartet (siehe Abschn. 2.1 auf S. 10). Um neben Linked Data-Anfragen anderer Knoten auch direkte Anfragen eines Browser an dieselben Ressourcen zu

ermöglichen, wird eine gezielte Auslieferung der Inhalte (*content negotiation*) durchgeführt. Xodx liest aus dem HTTP-Header der Anfrage das Feld `Accept`, welches in der Regel eine Reihe von MIME-Type-Angaben enthält und sucht die am ehesten passende Repräsentation der Ressource aus und leitet den anfragenden Client zu dieser weiter. Diese Prinzipien werden für alle Ressourcen angewendet, insbesondere dienen sie zur Umsetzung der Anforderungen „Kommunikation über Server- bzw. Anbietergrenzen hinweg“ in BA 3 auf S. 21 und „Beschreibungen anderer Personen abrufen“ in BA 5 auf S. 22.

5.4.2 Semantic Pingback

Semantic Pingback (siehe Abschn. 2.2 auf S. 10) wird im DSSN hauptsächlich in zwei Fällen eingesetzt. Einerseits zur Herstellung eines ersten Kontakts beim Aufbau einer Freundschaftsbeziehung und andererseits, um den Eigentümer über Aktivitäten zu benachrichtigen, die seine Ressourcen betreffen (Tramp et al., 2012). Diese beiden Fälle haben folgendes gemeinsam: wird auf einem Knoten eine Ressource veröffentlicht oder eine neue Relation hinzugefügt, die einen anderen Knoten interessieren könnte (z. B. ein Kommentar, eine Freundschafts-Relation oder Taggen einer Person auf einem Bild), muss dieser betreffende Knoten proaktiv darüber benachrichtigt werden. Im Folgenden werden beispielhaft Szenarien und die dazugehörigen Pings beschrieben.

Freundschaftsanfrage senden

Bei dem Aufbau einer neuen Freundschaftsbeziehung (engl. *friending* oder *befriending*) von einer Person zu einer anderen hat der Knoten mit der ausgehenden Relation Kenntnis über die zweite Person und den dazugehörigen Knoten, aber dieser nicht unbedingt über die Erste und deren Knoten. Um den Knoten mit der eingehenden Relation über diese neue Relation zu benachrichtigen, wird das Semantic Pingback-Protokoll eingesetzt. Dabei wird ein Ping mit der ersten Person als *source* und der zweiten Person als *target* (und einer optionalen nicht funktionalen Nachricht) gesendet.

Ein möglicher Ablauf einer Freundschaftsanfrage wird in Abb. 5.3 auf S. 43 dargestellt. Wenn zum Beispiel Amélie (`fb:?c=person&id=amelie`) Nino (`xo:?c=person&id=nino`) neu zu ihrer Freundesliste hinzufügt, dann kennt Nino zu diesem Zeitpunkt Amélie vielleicht noch nicht. Der Knoten von Amélie ruft die persönliche Beschreibung von Nino per Linked Data ab und ermittelt den angegebenen Semantic Pingback-Service (in diesem Fall `xo:?c=pingback&a=ping`). Ist dies gelungen, sendet der Knoten von Amélie einen Ping mit der *source* `fb:?c=person&id=amelie` und dem *tar-*

get `xo:?c=person&id=nino` an den Semantic Pingback-Service. Der Knoten von Nino kann nun entscheiden, ob und auf welche Weise er Nino darüber in Kenntnis setzt, z. B. durch Generieren einer Benachrichtigung (siehe Abschn. 5.3.5 auf S. 41).

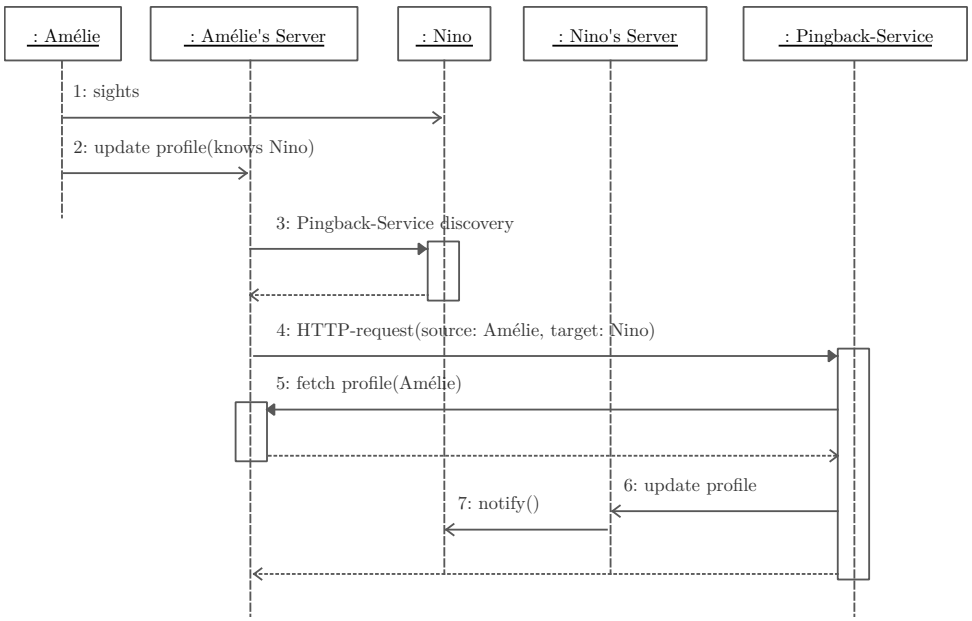


Abbildung 5.3: Sequenzdiagramm des Ablaufs einer Freundschaftsanfrage über Semantic Pingback

Der hier beschriebene Ablauf dient zur Umsetzung der Anforderungen „Kommunikation über Server- bzw. Anbietergrenzen hinweg“ in BA 3 auf S. 21 und „Freunde hinzufügen“ in BA 6 auf S. 22.

Einen Kommentar zu einer Ressource hinzufügen

Wenn eine Person eine Ressource auf einem anderen Knoten kommentiert, veröffentlicht sie die Kommentar-Ressource auf ihrem Knoten mit einem Verweis auf die kommentierte Ressource (Hinzufügen der Relation `sioc:reply_of`). Da ein Betrachter der kommentierten Ressource auch die dazugehörigen Kommentare lesen möchte, muss der Knoten Kenntnis von diesen erlangen, um einen Rückverweis einzutragen (Hinzufügen der Relation `sioc:has_reply`). Dies kann ebenfalls durch die Nutzung von Semantic Pingback umgesetzt werden, indem der kommentierende Knoten einen Ping mit der Kommentar-Ressource als *source* und der kommentierten Ressource als *target* sendet. Abbildung 5.4 auf S. 44 zeigt beispielhaft, wie ein Kommentar zu einer Ressource erstellt und an die entsprechenden Knoten weitergeleitet wird.

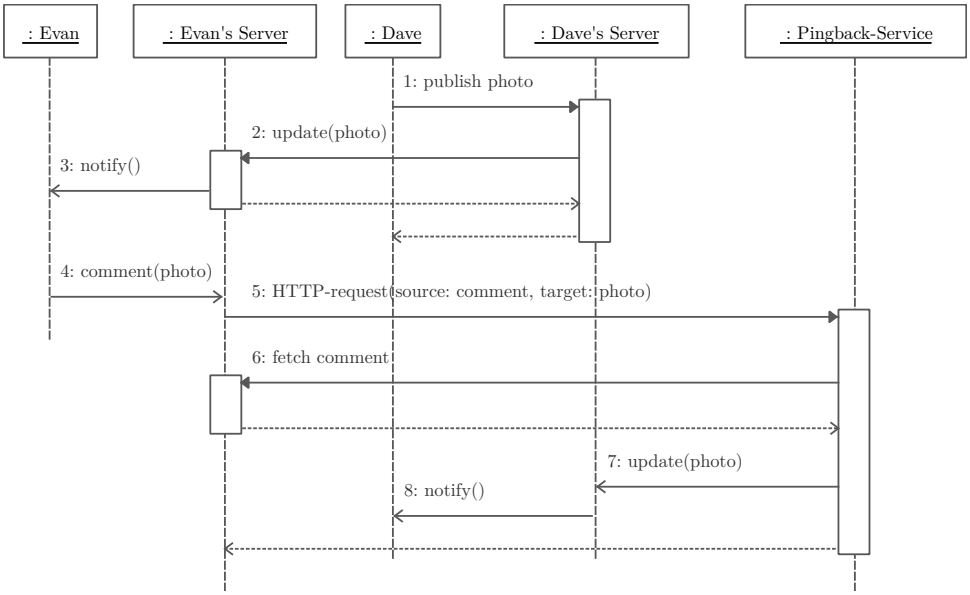


Abbildung 5.4: Sequenzdiagramm zum Kommentieren einer Ressource über Semantic Pingback

Der hier beschriebene Ablauf dient zur Umsetzung der Anforderungen „Kommunikation über Server- bzw. Anbietergrenzen hinweg“ in BA 3 auf S. 21 und „Ressourcen kommentieren“ in BA 12 auf S. 23.

Eingehender Semantic Pingback-Ping

Geht ein Semantic Pingback-Ping an einem Knoten ein, so sollte dieser daraufhin passende Aktionen durchführen. Nachdem der Ping gemäß dem Semantic Pingback-Protokoll als gültig befunden wurde, wird der Ping anhand der Relation, die *source* und *target* verbindet, klassifiziert.

Freundschaftsanfrage Handelt es sich um eine `foaf:knows`-Relation, so wird eine Benachrichtigung mit einer entsprechenden Botschaft für das Benutzerkonto, der als *target* angegebenen Person, angelegt.

Eingehender Kommentar Ist die Relation `sioc:reply_of`, so wird überprüft, ob die *source* vom Typ `sioct:Comment`, `aair:Comment`, `sioc:Post` oder `aair:Note` ist und das *target* vom Typ `sioc:Post` oder `aair:Note`. Ist dies der Fall, so wird dem *target* die Relation `sioc:has_reply` mit dem Kommentar als Objekt hinzugefügt.

Eingehender Tag auf einem Bild Ist die Relation `foaf:depicts`, die *source* vom Typ `foaf:Image` und das *target* vom Typ `foaf:Person`, so wird eine Benachrichtigung mit einer entsprechenden Botschaft für das Benutzerkonto, der als *target* angegebenen Person, angelegt. Ist die Relation `foaf:depiction`, die *source* vom Typ Person und das *target* vom Typ `foaf:Image`, so wird dem Bild die Relation `foaf:depicts` hinzugefügt.

Der hier beschriebene Ablauf dient zur Umsetzung der Anforderungen „Kommunikation über Server- bzw. Anbietergrenzen hinweg“ in BA 3 auf S. 21, „Freunde hinzufügen“ in BA 6 auf S. 22 „Medien-Artefakte veröffentlichen und Bilder taggen“ in BA 8 auf S. 22, „Ressourcen kommentieren“ in BA 12 auf S. 23 und „Benachrichtigungen“ in BA 13 auf S. 23.

5.4.3 Aktivitäten verteilen und empfangen mit PubSubHubbub

PubSubHubbub wird für Beinahe-Echtzeit-Verteilung und -Empfang von Aktivitäten eingesetzt. Alle Aktivitäten, die eine Ressource betreffen, sind in einem Activity-Stream, der als Atom-Feed vorliegt, zusammengefasst (Abschn. 5.3.3 auf S. 38). Werden zu dem Activity-Stream neue Aktivitäten hinzugefügt, so wird dieser an den PubSubHubbub-Hub (siehe Abschn. 2.3 auf S. 11) publiziert (*publish*) und alle Abonnenten des Streams durch den Hub darüber informiert. Der Knoten des Abonnenten erhält die neue Aktivität, liest den URI aus dem `id`-Feld und importiert die dazugehörige RDF-Ressource per Linked Data.

Dieser Ablauf dient zur Umsetzung der Anforderungen „Kommunikation über Server- bzw. Anbietergrenzen hinweg“ in BA 3 auf S. 21, „Statusnachrichten veröffentlichen“ in BA 7 auf S. 22, „Medien-Artefakte veröffentlichen und Bilder taggen“ in BA 8 auf S. 22, „Ressourcen abonnieren und Abonnementverwaltung“ in BA 10 auf S. 23, „Aktualisierungen zu abonnierten Ressourcen empfangen“ in BA 11 auf S. 23 und „Ressourcen kommentieren“ in BA 12 auf S. 23.

5.5 Service- und Anwendungs-Ebene

Die Service- und Anwendungs-Ebene dient zur Bereitstellung von Diensten oder Anwendungen, entweder für auf dem Knoten registrierte Personen oder für beliebige Personen im DSSN. Xodx stellt einen Profilmanager, ein System zum Teilen von Medien-Artefakten und ein Benachrichtigungssystem zur Verfügung. Zur Verwendung des PubSubHubbub-Protokolls nutzt Xodx einen externen frei wählbaren PubSubHubbub-Hub.

5.5.1 Profilmanager

Wenn eine Person mit Xodx am DSSN teilnehmen möchte, kann sie über den Profilmanager ein bereits bestehendes RDF-Dokument mit einer `foaf:Person` importieren. Es wird eine neue Person im Namespace der Xodx-Instanz angelegt, die die Eigenschaften und Relationen der importierten `foaf:Person` übernimmt. Ebenso bietet er die Möglichkeit, neue Freundschafts-Relationen hinzuzufügen. Mit Hilfe von OntoWiki (siehe Tramp et al. (2010)) können die gespeicherten Profildaten bearbeitet und ergänzt werden.

5.5.2 Teilen von Medien-Artefakten

In Xodx ist ein System zum Teilen von Medien-Artefakten enthalten. Es bietet einen Datei-Upload an, um Medien-Artefakte in das System zu laden. Der Benutzer kann durch hinzufügen einer `foaf:depicts`-Relation Personen, die auf einem Bild oder in einem Video zu sehen sind, taggen (siehe Abschn. 5.3.2 auf S. 36). Die Medien-Artefakte werden in den Activity-Stream des Teilenden eingetragen und sind per Linked Data verfügbar.

5.5.3 Benachrichtigungssystem

Das Benachrichtigungssystem kümmert sich darum, Personen über sie betreffende Aktionen im DSSN zu informieren. Momentan geschieht dies ausschließlich bei eingehenden Semantic Pingback-Pings (Abschn. 5.4.2 auf S. 44). Trifft ein solches Ereignis ein, erzeugt das Benachrichtigungssystem eine neue Benachrichtigungs-Ressource (Abschn. 5.3.5 auf S. 41), die dem Benutzer nach einem Login angezeigt wird.

5.6 Spezifikation der Systemanforderungen

Für die Verwaltung der semantischen Daten soll das Erfurt-Framework verwendet werden, da es schon viele notwendige Funktionen enthält. Auf Erfurt baut lib-dssn-php auf. In ihr sind bereits einige Funktionen zur Verwendung in einem DSSN-Knoten umgesetzt. Weitere Beschreibungen der Bibliotheken und deren Funktionsumfang sind in Abschn. 6.1.1 auf S. 48f zusammengefasst.

Da es sich bei Xodx um eine Webanwendung handelt, ist zur Ausführung ein Webserver notwendig. Getestet wurden Apache[2] und Nginx[3]. Xodx wurde in der Programmiersprache PHP („PHP: Hypertext Preprocessor“)[4] geschrieben, welche mindestens in der Version 5.3.7 vorausgesetzt wird. Als Webanwendungs-Framework wurde Saft (Abschn. 6.1.3 auf S. 49) entwickelt und in Kombination mit dem Autoloader des Zend-Frameworks[5] eingesetzt. Saft sollte in der aktuellen Version eingesetzt werden, Zend wurde in Version 1.12.0 getestet und wird in einer $1.1x$ Version benötigt. Zur Datenverwaltung und für einige Semantic Web-Funktionen kommen das Erfurt-Framework und die lib-dssn-php jeweils in der aktuellen Version zum Einsatz. Die Benutzerschnittstelle wird im Web-Browser dargestellt und nutzt das Bootstrap-Framework[6], getestet in Version 2.2.1–2.3.1, mit Abhängigkeiten zu jQuery, getestet in Version 1.8.2–1.9.1.

[2]The Apache HTTP Server Project: `http://httpd.apache.org/`
[3]nginx: `http://nginx.org/`
[4]„PHP: Hypertext Processor“: `http://www.php.net/`
[5]Zend-Framework: `http://framework.zend.com/`
[6]Bootstrap-Framework auf Github: `http://twitter.github.io/bootstrap/`

6 Implementierung

Im Rahmen dieser Arbeit wurde Xodx als Implementierung eines Knotens mit grundlegenden Funktionen zum Betrieb im DSSN entworfen und umgesetzt. Die Software wurde dabei von Grund auf neu entwickelt und setzt auf die Wiederverwendung bestehenden Programmcodes durch Bibliotheken. Nähere Informationen zum Bezug des Programm-Codes sind in Kap. 9 auf S. 68 zu finden.

6.1 Verwendete Bibliotheken und Frameworks

Xodx greift auf bestehende Bibliotheken und Frameworks zurück, die die Arbeit mit RDF-Daten und den verwendeten Protokollen erleichtern. Dies ermöglicht eine übersichtliche Struktur des Quellcodes und vermeidet unnötige Dopplungen. Im Folgenden wird der Funktionsumfang der eingesetzten Komponenten kurz erklärt. Zusätzlich zu den genannten Bibliotheken wird das Zend-Framework, sowohl für den Einsatz des Autoloaders als auch als Abhängigkeit des Erfurt-Frameworks, benötigt.

6.1.1 Erfurt-Framework

Erfurt[1] ist ein in PHP geschriebenes Semantic Web Framework, welches aus dem OntoWiki-Projekt[2] (siehe Tramp et al., 2010) entstanden ist und abstrahierten Zugriff auf einen Triple Store und dessen Modelle bereitstellt. Es können Daten in die Modelle eingefügt werden und mittels SPARQL/Update abgefragt und geändert werden. Zusätzlich bietet Erfurt weitere Funktionen zum Umgang mit RDF-Daten an. Xodx nutzt die SPARQL-Unterstützung zur Abfrage der Daten aus dem Modell. Mit der in Erfurt enthaltenen RDF-Serialisierung werden die Ressourcen bei einer Anfrage per Linked Data an Xodx in Turtle und RDF-XML serialisiert. Das Erfurt-Wrapper-System wird zum Import von RDF-Ressourcen nach den Linked Data-Prinzipien verwendet. Die Implementierung des Semantic Pingback-Protokolls wird zum Empfang von Pings verwendet.

[1] Erfurt Framework: `http://erfurt-framework.org/`
[2] OntoWiki: `http://ontowiki.net/`

6.1.2 lib-dssn-php

Ebenfalls aus dem OntoWiki-Projekt ist die PHP-Bibliothek lib-dssn-php[3] hervorgegangen. Sie stellt wichtige Funktionen zum Erstellen und Verarbeiten der Aktivitäten, des Activity-Streams und des -Feeds in einem DSSN zur Verfügung. Sie baut dazu direkt auf das Erfurt-Framework auf und kann eingehende Activity-Feeds parsen und in ein Erfurt-Modell importieren. Im Rahmen dieser Arbeit wurde sie um eine Funktion zur Verarbeitung von Aktivitäten erweitert.

6.1.3 Saft

Als Grundgerüst für Xodx wurde das Semantic Application Framework Saft[4] entwickelt und eingesetzt. Es ist für den Aufbau von Anwendungen gemäß dem *model-view-controller*-Entwurfsmuster (MVC) gedacht und bietet zu diesem Zweck ein Controller-System mit *actions* und ein templatebasiertes Layoutsystem an. Ein spezieller Controller ist der `ResourceController`, welcher jeweils für einen bestimmten Typ von RDF-Ressourcen passende *actions* anbietet (siehe Abschn. 6.2.1 auf S. 50). Mit dem Bootstrap-System können beliebige andere Ressourcen, u.a. das Daten-Modell, initialisiert werden. Außerdem werden einige andere häufig verwendete Funktionalitäten von Webanwendungen abstrahiert. Das Request-System bietet ein Anfrage-Objekt, über das Controller die Parameter der aktuellen HTTP-Anfrage abrufen können. Der Logger dient zur Protokollierung von beliebigen Aktionen und Fehlern im Betrieb und während der Entwicklung. Das Helper-System verwaltet Helper-Objekte, welche häufig vorkommende Funktionen zur Unterstützung der Controller bereitstellen. Da im DSSN viele unter Umständen zeitaufwändige HTTP-Anfragen zur Aktualisierung der Knoten untereinander durchgeführt werden, bietet Saft Worker zur asynchronen Ausführung von Aufgaben (*jobs*) an.

6.2 Organisation des Programmcodes

Als Application-Framework setzt Xodx Saft ein. Xodx ist gemäß dem MVC-Entwurfsmuster organisiert. Das Modell stellt die Daten-Ebene dar, speichert alle Benutzerdaten und ist im RDF formuliert in einer Virtuoso-Datenbank gespeichert. Der Zugriff auf die Datenbank erfolgt durch das Erfurt-Framework. Nähere Informationen zum Aufbau der Ressourcen in der Daten-Ebene sind in Abschn. 5.3 auf S. 35 zu finden. Die Controller sind als `Saft_Controller` implementiert und nehmen Anfragen entgegen, rufen die nötigen Daten aus dem Modell ab und geben sie mit

[3] lib-dssn-php: `https://github.com/AKSW/lib-dssn-php/`

[4] Semantic Application Framework Saft: `https://github.com/white-gecko/Saft`

der entsprechenden Ansicht aus. Die Controller sind dabei Teil der Protokoll-Ebene. Die Ansichten (*view*) werden durch das Saft-Layoutsystem mit Hilfe von Templates erzeugt. Funktionalität, die von mehreren Controllern benötigt wird, wird in Helperklassen ausgelagert.

6.2.1 Controller

Bei einer Anfrage an eine Xodx-Instanz müssen immer zwei Parameter angegeben werden. Der Paramter `c` gibt an, welcher Controller angefragt werden soll und der Parameter `a`, welche Aktion auf dem Controller ausgeführt werden soll. Der Name einer Controller-Klasse endet immer auf `Controller` und der Name der Funktionen, die eine Aktion bereitstellen, auf `Action`. Ist keine Aktion angegeben, wird automatisch die `indexAction` auf dem angegebenen Controller ausgeführt. Ist kein Controller angegeben, wird der `IndexController` angefragt.

Zur Implementierung von Controllern, die einen bestimmten Ressourcentyp verwalten, wurde der `ResourceController` vorgesehen. Diese Controller erweitern jeweils den `ResourceController`, welcher eine Unterklasse von `Saft_Controller` ist. Durch die Art der Anfragen an Controller wird auch der Aufbau der verwendeten URI von Ressourcen vorgegeben. Der Anfrage an den zuständigen Controller wird zusätzlich der Parameter `id` übergeben, um die entsprechende Ressource zu identifizieren.

ResourceController

Beliebige Ressourcen können mit dem `ResourceController` verwaltet werden. Er implementiert Linked Data mit *content negotiation* (vgl. Abschn. 5.4.1 auf S. 41) und leitet abhängig vom am besten passenden MIME-Type die Anfrage an die `rdfAction`, `showAction` oder `imgAction` weiter. Die `rdfAction` gibt eine Beschreibung der Ressource, serialisiert in Turtle oder RDF-XML, aus. Die `showAction` erzeugt eine HTML Beschreibung der Ressource. Die `imgAction` gibt, wenn es sich bei der Ressource um ein Bild handelt, eine Bilddatei aus. Ein URI einer allgemeinen Ressource sieht folgendermaßen aus: `fb:?c=resource&id=<id>`, wobei `<id>` durch eine zufällige Zeichenkette aus Zahlen und Buchstaben ersetzt wird, die bei der Erstellung der Ressource erzeugt wurde.

PersonController

Der `PersonController` ist zuständig für die Verwaltung von Personen. Er erweitert den `ResourceController` und überschreibt die `rdfAction` und die `showAction`. Die `rd-`

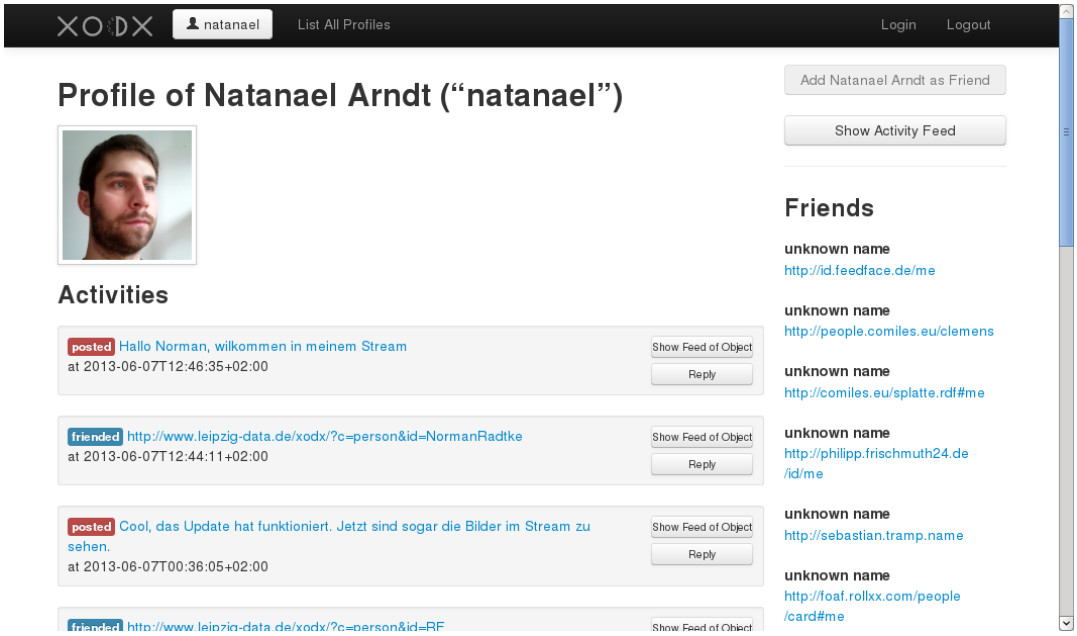


Abbildung 6.1: Ansicht einer Person in Xodx

`fAction` liefert eine RDF-Beschreibung in einem `foaf:PersonalProfileDocument` zurück und die `showAction` eine HTML-Repräsentation zur Anzeige im Browser (Abb. 6.1). Mit der `addfriendAction` kann durch einen autorisierten Agent ein neuer Freund zur Freundesliste einer Person hinzugefügt werden. Eine Person wird mit folgendem URI identifiziert: `fb:?c=person&id=<nickname>`, wobei `<nickname>` durch einen von der Person gewählten Spitznamen ersetzt wird.

MediaController

Der `MediaController` ist für die Verwaltung von Medien-Artefakten zuständig. Er erweitert ebenfalls den `ResourceController` und überschreibt die `showAction`, um die Medien-Artefakte in einer passenden Ansicht zu zeigen. Zusätzlich besitzt er die `imgAction`, mit der die Medien-Dateien direkt ausgegeben werden und die `tagAction`, mit der Personen auf Bildern markiert werden können. Abbildung 6.2 auf S. 52 zeigt die Ansicht der `showAction` mit der Bilddatei links, der Funktion zum Markieren von Personen rechts oben und dem Activity-Stream zu der Ressource darunter.

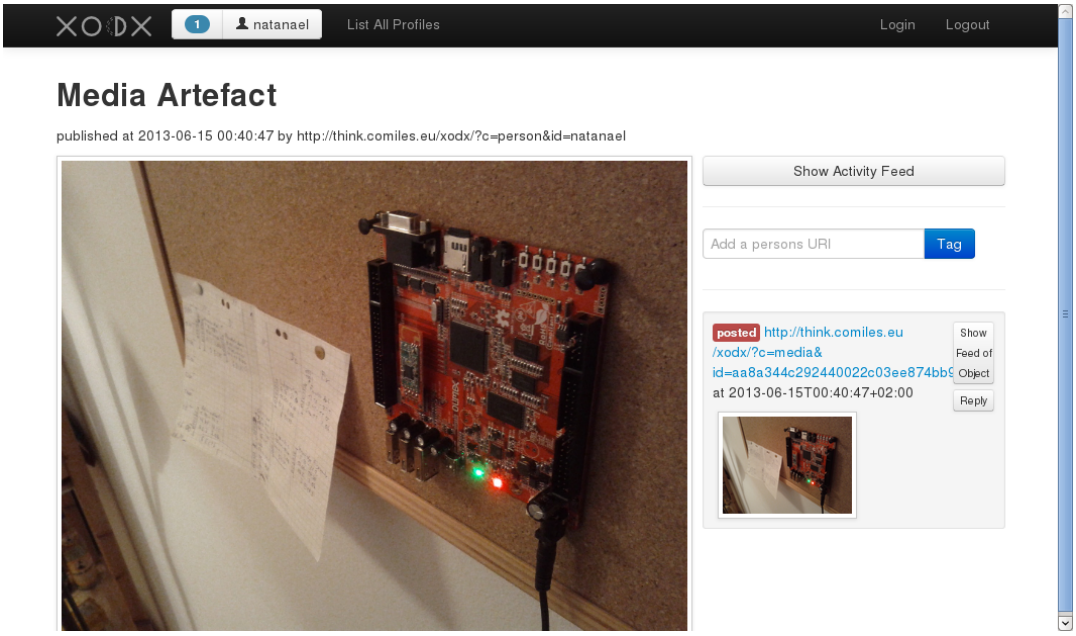


Abbildung 6.2: Ansicht eines Medien-Artefakts in Xodx

ActivityController

Der `ActivityController` verwaltet Aktivitäts-Ressourcen. Mit der `addactivityAction` können neue Aktivitäten einer Person zu ihrem Activity-Stream hinzugefügt werden. Die `replyAction` erzeugt eine Antwort-Ressource auf eine andere bestehende Ressource. Zum Einlesen externer Aktivitäten aus einem Activity-Feed wird die lib-dssn-php mit der `DSSN_Activity_Feed_Factory` eingesetzt.

PingbackController

Der `PingbackController` besitzt eine `pingAction`. Sie dient als Pingback-Service zum Empfang von Pings (vgl. Abschn. 2.2 auf S. 10) für Ressourcen, die durch die entsprechende Xodx-Instanz verwaltet werden. Bei einem eingehenden Ping für eine Ressource (*target*) wird versucht, das zuständige Benutzerkonto zu ermitteln, für das eine Benachrichtigung erzeugt werden soll. Dies geschieht nach folgenden Regeln:

1. Ist die Ressource vom Typ `foaf:Person` wird nach einem `sioc:UserAccount ?user` mit dem Triple `?user sioc:account_of <person>` gesucht.

2. Ist die Ressource vom Typ `aair:Activity` wird die Relation `aair:activityActor` gesucht und zu der dort gefundenen Person der entsprechende User gemäß Punkt 1 gesucht.

3. Ist die Ressource von einem anderen Typ wird gesucht ob sie in einer *activity* als Objekt referenziert ist. Dazu wird das Triple `?activity aair:activityObject <resource>` gesucht. Zu der daraus resultierenden *activity* wird der entsprechende User gemäß Punkt 2 gesucht.

Wurde am Ende eine Ressource vom Typ `sioc:UserAccount` gefunden, wird mit der `NotificationFactory` eine neue `dssn:Notification` erzeugt und dem entsprechenden Benutzerkonto zugeordnet. Konnte am Ende keine solche Ressource gefunden werden, wird der Ping in die Logdatei eingetragen und verworfen.

PushController

Der `PushController` verwaltet die Veröffentlichung der eigenen Feeds und das Abonnieren von fremden Feeds über das PubSubHubbub-Protokoll. Die `subscribe`-Methode bietet dazu anderen Controllern die Möglichkeit, einen Feed zu abonnieren. Während des Protokollablaufs ruft der Hub dann die `callbackAction` auf, um das Abonnement zu bestätigen. Beim Aufruf der Methode `publish` sendet der `PushController` die Information, dass ein Feed aktualisiert wurde und er alle Abonnenten benachrichtigen soll, an den Hub.

UserController

Der `UserController` bietet angemeldeten Benutzern die `homeAction` an (Abb. 6.3 auf S. 54). Sie stellt folgende Funktionen zur Verfügung: Anzeige neuer Benachrichtigungen, Activity-Stream mit den Aktivitäten aller abonnierter Ressourcen und die Verwaltung der Freundesliste. Der Controller besitzt zusätzlich die `getNotifications-Action`, welche ein JSON-Objekt mit den aktuellen Benachrichtigungen zurückliefert. Diese Funktion wird verwendet, um den Benutzer auf der Benutzeroberfläche über neue Benachrichtigungen zu informieren.

6.2.2 Layoutsystem

Xodx besitzt eine Benutzeroberfläche, die im Webbrowser angezeigt werden kann. Über diese Oberfläche kann ein Benutzer die einzelnen Controller bedienen, um sein Profil und andere Ressourcen zu erstellen, zu bearbeiten und anzuzeigen. Die Ausgabe

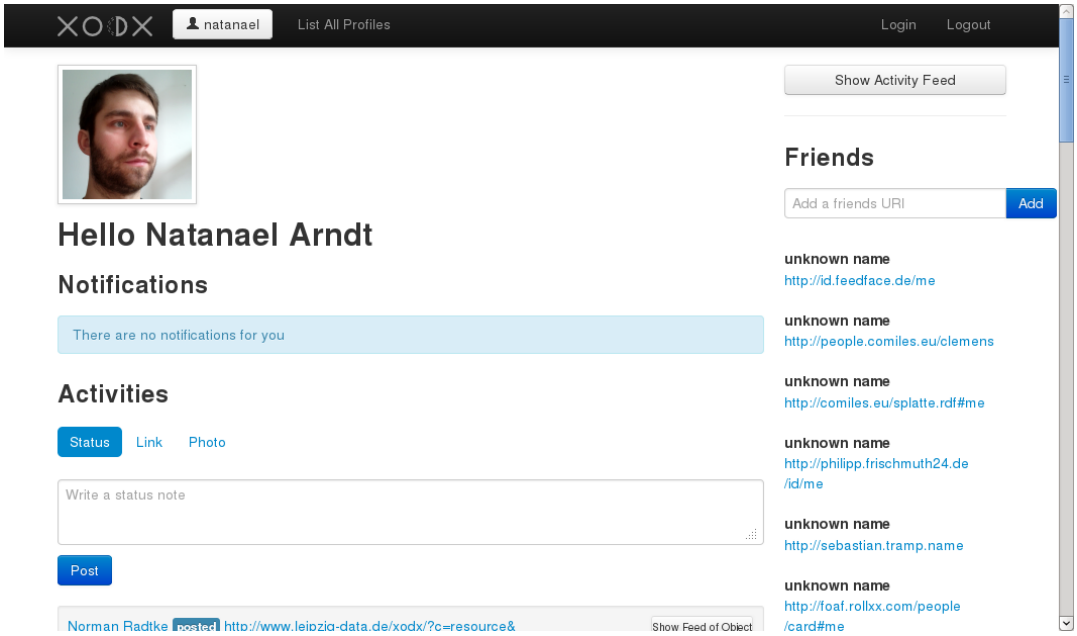


Abbildung 6.3: Dashboard-Ansicht im UserController in Xodx

wird aus einzelnen Vorlagen (*templates*) zusammengesetzt, die durch die Controller mit Werten gefüllt werden. Es gibt ein Grundgerüst (*layout template*) das auf allen Ansichten angezeigt wird und jeweils einen sich ändernden Inhalt, der von der aufgerufenen `Action` abhängig ist. Durch die Verwendung des Layoutsystems wird die Präsentation strikt von den Daten und der Funktionalität getrennt. Auf diese Weise können die Ausgaben in Zukunft auf beliebige andere Formate angepasst werden.

6.2.3 Helper

Die von mehreren Controllern benötigte Funktionalität wird in Helperklassen zusammengefasst. Diese Architektur dient dazu, die Funktionalität der Controller klar auf die Verarbeitung der Anfragen auszurichten und komplexeren Programmcode auszulagern. Eine Instanz eines Helpers kann von der aktuell laufenden Anwendung mit der Methode `getHelper` bezogen werden. Die Namen der Klassen enden immer auf `Helper`.

6.2.4 Benachrichtigungssystem

Das Benachrichtigungssystem dient dazu Personen über sie betreffende Aktionen im DSSN zu informieren. Wenn ein Controller eine Aktion verarbeitet, die für einen Benutzer relevant sein könnte, ruft er die Methode `forUser` auf der `NotificationFactory` auf. Diese nimmt als Parameter einen `User`, einen Text und optional einen `URI`, der auf die Ressource verweist, die für die Benachrichtigung relevant ist. Mit der Methode `fromModel` kann eine Benachrichtigungs-Ressource aus dem Modell gelesen werden.

7 Evaluation und Diskussion

Die in den vorhergehenden Kapiteln spezifizierte und implementierte Software muss auf ihre Eignung untersucht und ihre Möglichkeiten müssen diskutiert werden. Dazu wird zuerst die Integration des Knotens im gesamten DSSN betrachtet. Anschließend wird die Software auf die Erfüllung des „Social Web Acid Test — Level 0“ hin untersucht. Ein wichtiger Punkt aus der Anforderungsanalyse war „Privatsphäre, Datenschutz und Redefreiheit“ in BA 1 auf S. 20. Daher werden die Möglichkeiten zur Wahrung der Privatsphäre, des Datenschutzes und die Unterstützung der Redefreiheit mit der entstandenen Implementierung aufgezeigt. Im Rahmen eines Praktikums wurde die Software auf günstiger und stromsparender Hardware installiert, um die Nutzbarkeit in einem FreedomBox-Setup zu untersuchen. Die daraus gewonnenen Erkenntnisse werden hier dokumentiert. Die Evaluation wird durch einen Test der Software mit generierten Daten in einem Aufbau aus mehreren Knoten abgeschlossen.

7.1 Integration in das gesamte DSSN

Das DSSN ist ein über das komplette Internet verteiltes Netzwerk, das durch den Einsatz von grundlegenden und weit verbreiteten Web-Technologien stark in das bisherige Internet integriert ist. Die Knoten auf Basis der Xodx-Software sollen sich ebenfalls in das gesamte DSSN integrieren und mit beliebigen anderen DSSN-Knoten, die mit unterschiedlicher Software betrieben werden, transparent interagieren. Diese Integration wird durch den einheitlichen Einsatz der in „An Architecture of a Distributed Semantic Social Network“ (Tramp et al., 2012) spezifizierten Protokolle und Datenmodelle erreicht. Abbildung 5.2 auf S. 35 zeigt die Integration von Xodx mit anderen Knoten des DSSN und die Nutzung eines externen PubSubHubbub-Services.

Da ein Designgrundsatz des DSSN *service decoupling* ist (Tramp et al., 2012), soll auch jeder Xodx-Nutzer in der Lage sein, die von ihm genutzten Services frei zu wählen. Der PubSubHubbub-Service, über den die Activity-Streams veröffentlicht werden, ist dabei beliebig austausch- und konfigurierbar. Zur Veröffentlichung von Statusnachrichten, Medien-Artefakten und anderen Ressourcen kann der Benutzer

einen beliebigen Service nutzen. Anschließend kann er die erstellte Ressource manuell über den `ActivityController` zu seinem Activity-Stream hinzufügen und mit seinen Abonnenten teilen. Eine automatische Integration externer Services ist an dieser Stelle mit dem aktuellen Stand der Software nicht möglich, aber für eine kommende Version vorgesehen. Diese externen Services könnten sich gegenüber dem Xodx-Knoten authentifizieren und per *access delegation* auf einen Update-Service zugreifen (Abschn. 8.1.2 auf S. 66). Ein alternativer Semantic Pingback-Service kann vom Nutzer eingestellt werden. Allerdings ist dann, ebenfalls aufgrund des fehlenden Update-Services, keine Benachrichtigung über Xodx möglich. Der Semantic Pingback-Service kann dem Benutzer allerdings bei eingehenden Pings eine E-Mail senden.

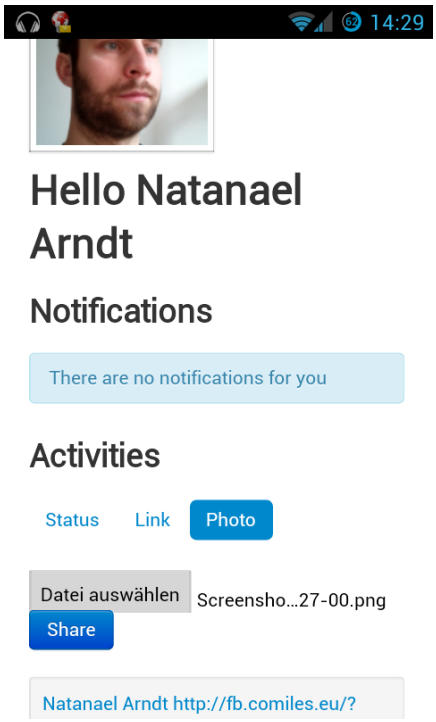
7.2 Social Web Acid Test

Der „Social Web Acid Test — Level 0“ (SWAT0)[1] ist ein Integrationstest, der grundlegende Funktionalitäten eines verteilten Online Social Networks abdeckt. SWAT0 wurde ebenfalls im Anwendungsfall 1: „Foto teilen, taggen und kommentieren“ (S. 15) aufgegriffen und für die Anforderungsanalyse verwendet. Tramp et al. (2012) haben bereits in „An Architecture of a Distributed Semantic Social Network“ gezeigt, dass die Architektur des DSSN den Test erfüllen kann.

1. With his phone, Dave takes a photo of Tantek and uploads it using a service
2. Dave tags the photo with Tantek
3. Tantek gets a notification on another service that he's been tagged in a photo
4. Evan, who is subscribed to Dave, sees the photo on yet another service
5. Evan comments on the photo
6. David and Tantek receive notifications that Evan has commented on the photo

Der erste Punkt fordert einen Zugang über mobile Endgeräte um ein Foto aufzunehmen und in dem Online Social Network zu teilen. Abbildung 7.1a auf S. 58 zeigt die Dashboardansicht mit der Funktion zum Teilen eines Fotos auf einem Android-Mobiltelefon. Mit dem `MediaController` (Abschn. 6.2.1 auf S. 51), wie in Abb. 7.1b auf S. 58 auf einem Mobiltelefon zu sehen, kann Dave anschließend Tantek auf dem geteilten Foto markieren. Tanteks Service wird per Semantic Pingback (Abschn. 6.2.1 auf S. 52) darüber informiert, welcher wiederum für Tantek eine Benachrichtigung erzeugt (Abschn. 6.2.4 auf S. 55). Da Evan den Activity-Stream von Dave abonniert hat, wird sein Service von Daves Service per PubSubHubbub (Abschn. 6.2.1 auf S. 53) aktualisiert und zeigt das Bild in Evans Dashboardansicht an. Evan kann nun mit dem `ActivityController` Abschn. 6.2.1 auf S. 52 eine Antwortressource erzeugen und einen Kommentar verfassen. Daves Service wird per Semantic Pingback über Evans

[1] Social Web Acid Test - Level 0: `http://www.w3.org/2005/Incubator/federatedsocialweb/wiki/SWAT0`



(a) Dashboard-Ansicht von Xodx auf einem Android-Mobiltelefon

(b) Medien-Artefakt-Ansicht von Xodx auf einem Android-Mobiltelefon

Abbildung 7.1: Ansichten zum Teilen und Taggen eines Fotos auf einem Android-Mobiltelefon

Antwortressource benachrichtigt und fügt sie in den Activity-Stream des Fotos ein. Wenn Tantek diesen Activity-Stream auf die Benachrichtigung hin abonniert hat, erhält auch er den Kommentar von Evan.

7.3 Privatsphäre, Datenschutz und Redefreiheit im DSSN

In der momentanen Xodx-Implementierung sind alle Aktivitäten und Ressourcen öffentlich abrufbar. Dies lässt Einwände bezüglich der Privatsphäre und dem Datenschutz zu, da der Benutzer keine Möglichkeiten hat, diese Einstellungen anzupassen. Im DSSN ist daher zusätzlich ein authentifizierter Zugriff auf Ressourcen vorgesehen, auf dessen Basis Zugriffsrechte überprüft und durchgesetzt werden können. Die Zugriffsrechte können z. B. mit Hilfe von WebAccessControl[2] formuliert werden. Durch Nutzung von WebID/FOAF+SSL (Sporny et al., 2011) können sich Agenten mit

[2]WebAccessControl: `http://www.w3.org/wiki/WebAccessControl`

Hilfe von Zertifikaten authentifizieren. Der Knoten kann daraufhin entsprechend der Zugriffsrechte Zugriff auf die Daten gewähren.

Ein Problem stellen allerdings immer noch die Activity-Feeds dar, die über PubSubHubbub veröffentlicht werden. Der Nutzer vertraut nicht unbedingt dem PubSubHubbub-Hub, um ihm private Daten zu überlassen. Ein Vertrauen wäre auch nicht angebracht, da die Nachrichten unverschlüsselt über den Hub übertragen werden. Ein Authentifizierung von Seiten des PubSubHubbub-Hubs ist nicht vorgesehen und somit muss der Feed öffentlich zugänglich sein. Es bieten sich an dieser Stelle zwei Möglichkeiten an. Zum einen könnte das PubSubHubbub-Protokoll um die nötigen Funktionen erweitert werden, die eine private Übertragung der Feeds ermöglichen. Zum anderen könnte nur die öffentliche Kommunikation über das PubSubHubbub-Protokoll abgewickelt werden und private Daten direkt zwischen den Knoten ausgetauscht werden.

Durch die Installation von Xodx auf einem Webserver mit HTTPS und Festplattenverschlüsselung kann der Datenschutz und der Schutz der Privatsphäre zusätzlich erhöht werden. Da es sich beim DSSN um ein verteiltes Netzwerk handelt kann die Redefreiheit nicht an einem zentralen, alle Nutzer betreffenden, Punkt eingeschränkt werden. Die Angreifbarkeit des Netzes bezüglich der Redefreiheit hängt vom Verteilungsgrad der Knoten ab. Der Verteilungsgrad kann durch den Einsatz von einem eigenen Knoten pro Teilnehmer erhöht werden. Dies ist die Vision des FreedomBox-Projekts und wird im folgenden Abschnitt näher untersucht.

7.4 FreedomBox

Im Wintersemester 2012/2013 wurde im Rahmen des Praktikums „Distributed Semantic Social Networks“ der Aufbau eines DSSN mit Hilfe mehrerer leistungsschwacher Computer untersucht. Ziel war dabei, ähnlich der Idee des FreedomBox-Projekts[3], einen Aufbau zu erhalten, bei dem jeder Teilnehmer des Netzes einen eigenen kostengünstigen und energieeffizienten Computer besitzt. Auf diesem Computer kann jeder Teilnehmer seine privaten Daten verwalten und veröffentlichen.

Die für das Praktikum verwendete Test-Hardware war ein A13-OLinuXino-WIFI-Board von Olimex mit einem 1GHz ARMv7 Cortex A8 Prozessor (Allwinner A13 *system-on-a-chip*), 512 MB RAM und 2–8 GB SD-Karten Speicher. Als Betriebssystem wurde ein für das Board angepasstes Debian GNU/Linux (arm hard float

[3]FreedomBox ist die Vision und Idee eine Hardware- und Softwareplattform zur freien Kommunikation zwischen unterschiedlichen Personen zu entwickeln.
FreedomBox Stiftung: `https://freedomboxfoundation.org/`,
FreedomBox im Debian Wiki: `https://wiki.debian.org/FreedomBox`.

port, armhf) installiert. Der Linux-Kernel wurde aus den Quellen des Linux-Sunxi Projekts[4] in der Version 3.0.42 kompiliert. Zum Betrieb der Xodx-Software wurde der leichtgewichtige und ressourcenschonende Web-Server NGINX verwendet. Der PHP-Code wurde mit der fast-cgi-Erweiterung ausgeführt. Als Datenbank wurde OpenLink Virtuoso Open-Source Edition[5] verwendet, da sie am besten durch das Erfurt-Framework unterstützt wird.

Durch das Praktikum konnte gezeigt werden, dass die Xodx-Software ohne spezielle Anpassungen auf solcher Hardware lauffähig ist. Es kann also mit Hilfe von Xodx ein verteiltes Online Social Network auf kostengünstiger und energieeffizienter Hardware aufgebaut werden. In diesem Netzwerk behält jeder Teilnehmer neben der Kontrolle über die Daten auch die volle Kontrolle über die Hardware.

Angeregt durch dieses Praktikum wurde in einem späteren Praktikum ebenfalls eine Installation der Software auf leistungsfähigerer Hardware vorgenommen. Diese Installation wird testweise im Rahmen der „Leipziger Initiative für Offene Daten“[6] zum Betrieb eines Knotens[7] eingesetzt.

7.5 Testaufbau eines Social Networks

Um das Zusammenspiel der Software mit anderen Knoten zu untersuchen wurde ein Integrationstest entworfen. Dieser Test deckt dabei die Anforderungen „Kommunikation über Server- bzw. Anbietergrenzen hinweg“ in BA 3 auf S. 21, „Persönliche Beschreibung anlegen und bearbeiten“ in BA 4 auf S. 22, „Freunde hinzufügen“ in BA 6 auf S. 22, „Statusnachrichten veröffentlichen“ in BA 7 auf S. 22, „Ressourcen abonnieren und Abonnementverwaltung“ in BA 10 auf S. 23, „Aktualisierungen zu abonnierten Ressourcen empfangen“ in BA 11 auf S. 23 und „Ressourcen kommentieren“ in BA 12 auf S. 23 ab.

Für den Integrationstest wurde eine Testkonfiguration mit 20 Knoten (siehe Abb. 7.2 auf S. 61) erstellt. Auf jedem Knoten ist die Xodx-Software mit einer Virtuso Open-Source-Datenbank installiert. Um die Personen auf den Knoten zu simulieren, wurde ein Agent entwickelt, der die Funktionen `signup`, `login`, `addFriend`, `post`, `reply` und `getActivities` zur Kommunikation mit anderen Agenten bereit stellt. Zusätzlich wurde die Funktion `stats` entwickelt, um statistische Informationen des Knotens auszulesen. Die Funktionen greifen direkt auf die entsprechenden Actions der Xodx-Software zu.

[4] Linux-Sunxi project wiki: `http://linux-sunxi.org`,
Kernel Sources: `https://github.com/linux-sunxi/linux-sunxi`
[5] OpenLink Virtuoso: `http://ods.openlinksw.com/dataspace/doc/dav/wiki/Main/Main.VOSIndex`
[6] http://www.leipzig-data.de/
[7] `http://www.leipzig-data.de/xodx/`

Jedem Knoten ist ein Agent zugeordnet. Der Agent erstellt ein Benutzerkonto mit einer persönlichen Beschreibung und erstellt Freundschaftsverknüpfungen zu anderen zufällig ausgewählten Agenten. Während des Aufbaus der Freundschaftsverknüpfungen abonniert er die Activity-Streams der Freunde.

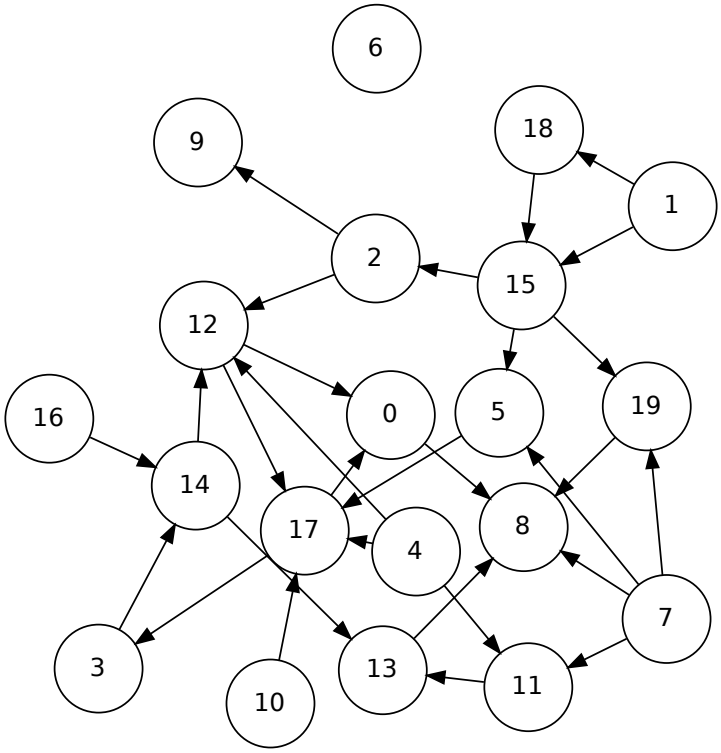


Abbildung 7.2: Darstellung der Testkonfiguration mit 20 Knoten und 29 Freundschaftsverknüpfungen. Die Pfeilspitzen zeigen in Richtung des Datenflusses.

Die Simulation wird durch das Objekt `testbed` gesteuert und läuft rundenweise ab. Um unterschiedliches Nutzerverhalten abbilden zu können, hat jeder Agent zusätzlich die Eigenschaften `responsivity`, `productivity` und `updateprobability`. Die Eigenschaft `productivity` gibt an, wie wahrscheinlich es ist, dass der Agent in einer Runde eine neue Nachricht generiert und veröffentlicht. Die `updateprobability` gibt an, wie wahrscheinlich der Agent in einer Runde seinen Activity-Stream aktualisiert, um Aktivitäten seiner Freunde zu empfangen. Wie wahrscheinlich der Agent auf eine neue Aktivität reagiert, wird durch die Eigenschaft `responsivity` angegeben.

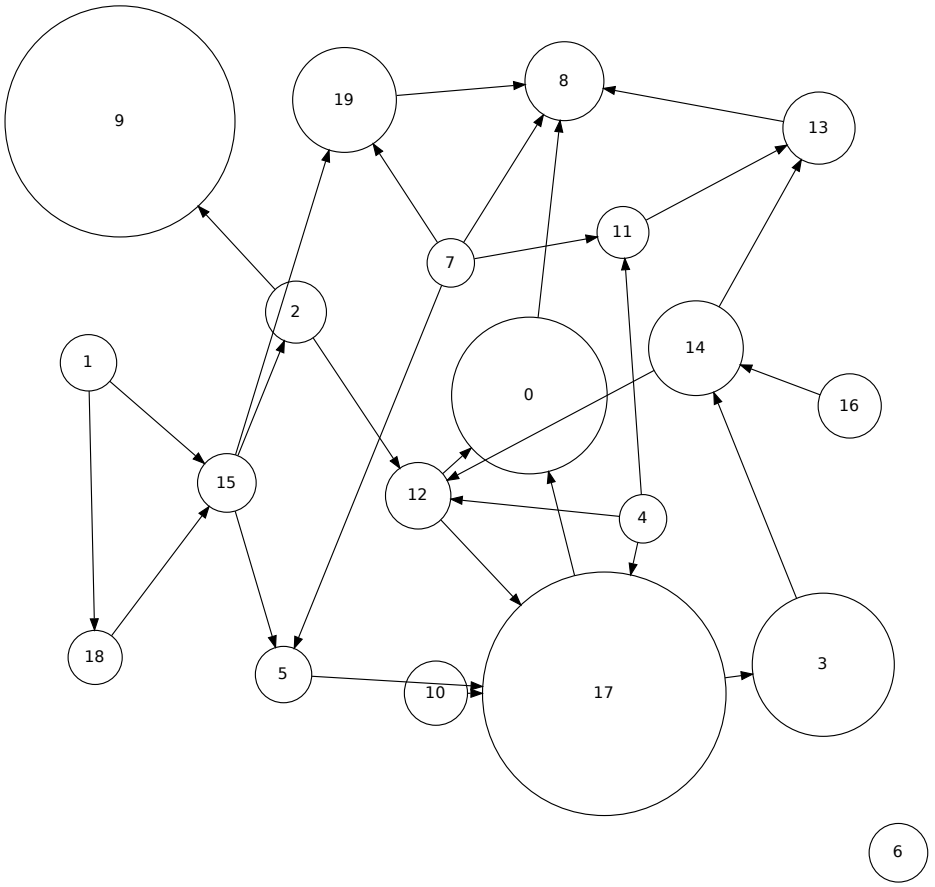


Abbildung 7.3: Darstellung der Testkonfiguration. Die Größe der Knoten gibt die Anzahl der entstandenen *activities* nach 10 Runden an. Die Pfeilspitzen zeigen in Richtung des Datenflusses.

In jeder Runde werden auf jedem Knoten die entstandenen RDF-Triple gezählt. Abbildung 7.3 auf S. 62 stellt das Netzwerk nach 10 Runden dar. Dabei wurden insgesamt 328 *activities* und 5911 Triple erzeugt. Besonders fallen die Knoten 9 (mit 71 *activities*) und 17 (mit 76 *activities*) auf. Der Agent auf Knoten 9 hat zwar nur Knoten 2 abonniert, dafür aber mit der Konfiguration `updateprobability = 1.0`, `responsivity = 0.5` und `productivity = 0.3` in jeder Runde die empfangenen Updates mit einer Wahrscheinlichkeit von $\frac{1}{2}$ kommentiert. Dabei kann es aufgrund der Implementierung des Agenten auch dazu gekommen sein, dass er dieselbe Nachricht in mehreren Runden kommentiert hat. Der Agent auf Knoten 17 hat mit einer `productivity` von `0.6` in jeder Runde wahrscheinlich eine neue Nachricht generiert. Hinzu kommt, dass er 4 anderen Knoten gefolgt ist und jede ihrer Aktualisierungen ebenfalls empfangen hat.

Einen weiteren interessanten Fall stellt Knoten 8 (Abb. 7.3) mit nur 12 *activities* dar. Der Agent auf Knoten 8 folgt 4 anderen Agenten (0, 7, 13 und 19), trotzdem ist Knoten 8 kleiner als Knoten 0 (mit 42 *activities*) und Knoten 19 (mit 22 *activities*). Hier kommt eine weitere Eigenschaft des Netzwerkes zum tragen. Jeder Knoten erhält nur Aktualisierungen zu Nachrichten, die von Agenten erzeugt wurden, denen sie direkt folgen. Knoten 0 und 19 enthalten also hauptsächlich Nachrichten ihrer Freunde, die demzufolge nicht an Knoten 8 weitergegeben wurden. Mit der Konfiguration `productivity = 0.2` für Knoten 0 ist dies auch sofort einleuchtend. Knoten 19 hat aber mit `productivity = 0.8` einen relativ hohen Wert. Die selbst generierten Nachrichten fallen gegenüber den Antworten auf bestehende *activities* nicht so sehr ins Gewicht, da jeder Knoten maximal eine Nachricht pro Runde erzeugt.

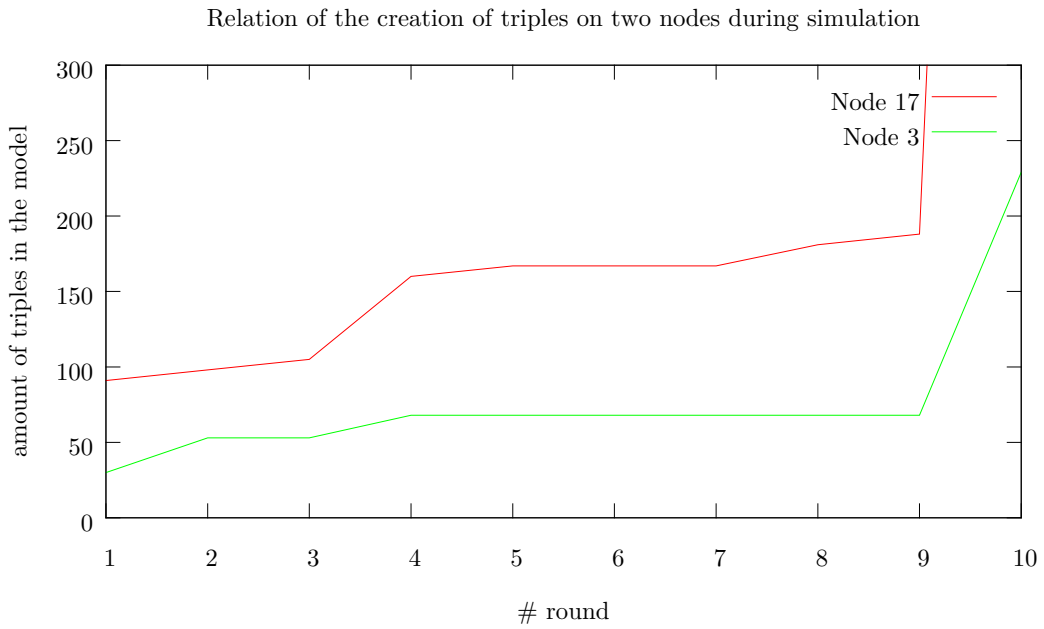


Abbildung 7.4: Entwicklung der Anzahl der Triple auf Knoten 17 und 3. Der Agent auf Knoten 3 folgt dem Agenten auf Knoten 17. Die Anzahl der Triple wurde immer zum Ende einer Runde gemessen.

Der Agent auf Knoten 3 folgt dem Agenten auf Knoten 17. Diese Beziehung lässt sich auch in Abb. 7.4 erkennen, da die Anzahl der Triple auf Knoten 3 abhängig von der Anzahl der Triple auf Knoten 17 ist. Da nur eine Messung der Anzahl der Triple stattgefunden hat und nicht die genaue Entstehung der Ressourcen über den Zeitraum der Simulation überwacht wurde, können die Interaktionen nicht vollständig nachvollzogen werden. Dennoch wird dieser Zusammenhang in Runde 4 das erste mal deutlich. Knoten 17 erstellt 4 *activities* und die dazugehörigen Post- und Comment-Ressourcen, dies sind 56 Triple, und Knoten 3 empfängt die 4 *activities* (28 Triple). Der Graph steigt entsprechend der Anzahl der Triple zwischen Runde 3 und 4. Ein

weiteres Mal erzeugt Knoten 17 in Runde 10 *activities* und auch der Graph von Knoten 3 steigt rapide an.

Mit diesem Versuch konnte gezeigt werden, dass mehrere Knoten mit der entstandenen Software betrieben werden und untereinander erfolgreich Nachrichten austauschen können. Durch eine genauere Justierung der einzelnen Parameter und einer Verbesserung des Agenten können weitere Versuche unternommen werden. Durch den Einsatz leistungsfähiger Hardware oder einer Simulation auf mehreren FreedomBoxes könnten auf diese Weise auch größere Netzwerke untersucht werden.

8 Zusammenfassung und Ausblick

Die momentan verfügbaren Möglichkeiten zur Kommunikation in Online Social Networks sind vor allem bezüglich des Datenschutzes und der Privatsphäre mangelhaft (vgl. Abschn. 1.1 auf S. 7). Daher hat sich diese Arbeit zum Ziel gesetzt einen verteilten Ansatz zum Aufbau eines Online Social Networks zu finden und einen entsprechenden Netzwerkknoten zu implementierten (vgl. Abschn. 1.2 auf S. 8). Es wurden zum erreichen dieses Zieles Ansätze auf Basis von Semantic Web-Technologien als passend befunden. Sie sind flexibel und bieten viele Erweiterungsmöglichkeiten. Tramp et al. (2012) haben eine solches verteiltes Online Social Network unter Verwendung von Semantic Web-Technologien, das DSSN, entwickelt.

Zur Realisierung einer solchen Software zum Betrieb eines Knotens im DSSN wurden Anwendungsfälle und die daraus resultierenden Anforderungen formuliert und zusammengetragen (vgl. Kap. 3 auf S. 14). Sie umfassen die wichtigsten Funktionen zur Kommunikation in einem Online Social Network und können durch den Einsatz zusätzlicher Services erweitert werden. Auf Basis der Anforderungen wurde ein Entwurf der Software erarbeitet (vgl. Kap. 5 auf S. 31). Sie setzt zur Datenhaltung und -repräsentation semantische Technologien ein (vgl. Abschn. 5.3 auf S. 35). Zur Kommunikation mit anderen Knoten des Netzwerks werden die Protokolle Linked Data, Semantic Pingback und PubSubHubbub mit Activity-Streams verwendet (vgl. Abschn. 5.4 auf S. 41). Die Software bietet dem Benutzer Funktionen zur Erstellung einer persönlichen Beschreibung, zur Verwaltung von Freundschaftsbeziehungen und zur Kommunikation mit anderen Teilnehmern des Netzwerkes an (vgl. Abschn. 5.5 auf S. 45). Gemäß dieser Spezifikation wurde die Software anschließend umgesetzt (vgl. Kap. 6 auf S. 48).

Es wurde untersucht, ob die Software ihren Anforderungen entspricht und mit ihrer Hilfe ein Distributed Semantic Social Network aufgebaut werden kann (vgl. Kap. 7 auf S. 56). Dazu wurde unter anderem die Software im Rahmen eines Praktikums auf leistungsschwacher, kostengünstiger und energieeffizienter Hardware installiert und auf diese Weise eine FreedomBox umgesetzt (vgl. Abschn. 7.4 auf S. 59). Außerdem wurde ein Testaufbau aus mehreren Knoten erstellt. In diesem Testaufbau haben Agenten automatisch generierte Nachrichten ausgetauscht. Auf diese Weise konnte auch die Kommunikation zwischen Knoten und deren Skalierungseigenschaften untersucht werden (vgl. Abschn. 7.5 auf S. 60).

Die entstandene Software bietet alle grundlegenden Funktionen, die für eine Kommunikation zwischen Teilnehmern auf unterschiedlichen Servern nötig ist. Sie wird auch bereits testweise im Rahmen der „Leipziger Initiative für Offene Daten“[1] zum Betrieb eines Knotens[2] eingesetzt. Aufgrund der flexiblen Architektur können auch andere Technologien im Internet, die auf den gleichen Prinzipien, der Verteiltheit und maschinenlesbaren strukturierten Datenrepräsentation beruhen, umgesetzt werden. Wenn dieser Ansatz weiter verfolgt wird können Benutzer von Online Social Networks bald selbst über die Verwendung ihrer Daten entscheiden.

8.1 Erweiterungsmöglichkeiten

Während des Entwurfs und der Entwicklung der Software haben sich einige Möglichkeiten gezeigt, wie die Xodx-Software erweitert werden kann. Dies ist einerseits die Implementierung weiterer Funktionen, die bereits teil der Architektur des DSSN sind und andererseits neue Funktionen, die von der Architektur noch nicht abgedeckt werden aber für einen produktiven Einsatz nötig sind.

8.1.1 WebID/FOAF+SSL und WebACL

Um private Kommunikation zwischen Personen auf unterschiedlichen Knoten zu ermöglichen muss eine verschlüsselte und authentifizierte Kommunikation zwischen den Knoten sichergestellt werden. Der entstehende WebID-Standard (Sporny et al., 2011) bietet ein Authentifizierungssystem an, das dazu verwendet werden kann. Mit Hilfe von Zertifikaten kann sich so ein Agent (eine Person oder ein anderer Knoten) gegenüber einem Knoten authentifizieren. Der Benutzer kann dann mit WebAccessControl[3] Zugriffsregeln für seine Daten definieren. Auf Basis dieser Regeln und der Authentifizierung kann der Knoten dann entsprechend Zugriff gewähren oder verweigern.

8.1.2 Update-Service

Um externen Anwendungen und Services die Möglichkeit zur Integration mit anderen DSSN-Knoten zu gewähren ist ein Update-Service vorgesehen. Für den schreibenden Zugriff auf andere Semantic Web-Anwendungen hat sich SPARQL (The-W3C-SPARQL-Working-Group, 2013) etabliert. Der Zugriff per SPARQL kann mit

[1] http://www.leipzig-data.de/
[2] `http://www.leipzig-data.de/xodx/`
[3] WebAccessControl: `http://www.w3.org/wiki/WebAccessControl`

Hilfe von WebID per *access delegation* reguliert werden. Durch Bereitstellung eines SPARQL-Endpoints kann so ein Update-Service realisiert werden.

8.1.3 Persönliche Nachrichten

In Online Social Networks ist es oft möglich, auch direkt Nachrichten zwischen zwei Teilnehmern auszutauschen, ohne dass Dritte den Inhalt der Nachrichten sehen oder von dem Vorgang Kenntnis erlangen. Eine solche Funktion ist noch nicht in der Architektur des DSSN vorgesehen, kann aber als Anwendung ergänzt werden.

Eine möglicher Ansatz ist durch Einbindung bestehender Instant-Messaging-Systeme wie etwa Jabber/XMPP oder E-Mail eine solche Funktion bereitzustellen. Alternativ kann ein neues System mit Hilfe von WebID/FOAF+SSL implementiert werden, bei dem der absendende Dienst sich dem empfangenden Dienst gegenüber authentifiziert und vergleichbar mit dem Versenden einer E-Mail, eine Nachricht für den Empfänger auf dessen Server hinterlegt. Dies könnte über den Update-Service geschehen.

9 Quellcode der Software

Der Quellcode zu dem beschriebenen System wurde als OpenSource-Projekt entwickelt und steht zur Weiterentwicklung zur Verfügung. Die Projekt Seite ist unter `http://xodx.dssn.org` erreichbar und der Quellcode ist auf der Github-Projektseite unter `https://github.com/white-gecko/xodx` abrufbar. Die aktuelle Entwicklungsversion des Quellcodes kann mittels Git[1] von `git://github.com/white-gecko/xodx.git` heruntergeladen[2] werden.

[1] Git Source Control Management: `http://git-scm.com/`

[2] Hilfestellungen zum Herunterladen eines Git-Repositories: `http://git-scm.com/book/en/Git-Basics-Getting-a-Git-Repository#Cloning-an-Existing-Repository`

Literaturverzeichnis

Arndt, N. (2010a). Entwicklung eines mobilen Social Semantic Web Clients. In Fähnrich, K.-P. & Franczyk, B. (Eds.), *INFORMATIK 2010: Service Science – Neue Perspektiven für die Informatik*, volume 2 of *GI-Edition – Lecture Notes in Informatics*, (pp. 1004–1005). Gesellschaft für Informatik e.V.

Arndt, N. (2010b). Entwicklung eines mobilen Social Semantic Web Clients. Bachelor's thesis, Universität Leipzig, Fakultät für Mathematik und Informatik, Institut für Informatik.

Arndt, N. (2011). TriplePlace: A flexible triple store for Android with six indices. In S. Auer, T. Riechert, & J. Schmidt (Eds.), *SKIL 2011 - Studentenkonferenz Informatik Leipzig 2011*, volume 27 of *Leipziger Beiträge zur Informatik* (pp. 1–7). Leipzig: LIV.

Atkins, M., Norris, W., Messina, C., Wilkinson, M., & Dolin, R. (2011). Atom Activity Streams 1.0. `http://activitystrea.ms/specs/atom/1.0/`.

Berners-Lee, T. (2009). Linked Data. Design issues, W3C. `http://www.w3.org/DesignIssues/LinkedData.html`.

Berners-Lee, T., Cailliau, R., Groff, J.-F., & Pollermann, B. (1992). World-Wide Web: The Information Universe. *Electronic Networking: Research, Applications and Policy*, *2*(1), 52–58.

Berners-Lee, T., Fielding, R., & Masinter, L. (2005). Uniform Resource Identifier (URI): Generic Syntax. RFC 3986, The Internet Engineering Task Force (IETF). `http://tools.ietf.org/html/rfc3986`.

Breslin, J. G., Harth, A., Bojars, U., & Decker, S. (2005). Towards Semantically-Interlinked Online Communities. In Gomez-Perez, A. & Euzenat, J. (Eds.), *European Semantic Web Conference (ESWC)*, volume 3532 of *Lecture Notes on Computer Science*, (pp. 500–514). Springer.

Brickley, D. & Miller, L. (2004). FOAF Vocabulary Specification. Namespace Document 2 Sept 2004, FOAF Project. `http://xmlns.com/foaf/0.1/`.

Dooley, K. (2001). *Designing large-scale LANs*. O'Reilly Media.

Graffi, K., Groß, C., Stingl, D., Hartung, D., Kovacevic, A., & Steinmetz, R. (2011). LifeSocial.KOM: A Secure and P2P-based Solution for Online Social Networks. In *Proceedings of the IEEE Consumer Communications and Networking Conference*.

Hollander, D., Bray, T., & Layman, A. (1999). Namespaces in XML. W3C Recommendation, W3C. http://www.w3.org/TR/1999/REC-xml-names-19990114.

Klyne, G. & Carroll, J. J. (2004). Resource Description Framework (RDF): Concepts and Abstract Syntax. W3C Recommendation, W3C. http://www.w3.org/TR/2004/REC-rdf-concepts-20040210.

Langridge, S. & Hickson, I. (2002). Pingback 1.0. Technical report. http://hixie.ch/specs/pingback/pingback.

Lassila, O. & Swick, R. R. (1999). Resource Description Framework (RDF) Model and Syntax Specification. W3C Recommendation, W3C. http://www.w3.org/TR/1999/REC-rdf-syntax-19990222.

Minno, M. & Palmisano, D. (2010). Atom Activity Streams RDF mapping. Technical report, NoTube Project. http://xmlns.notu.be/aair/.

Nottingham, M. & Sayre, R. (2005). The Atom Syndication Format. RFC 4287, The Internet Engineering Task Force (IETF). http://tools.ietf.org/html/rfc4287.

Radtke, N. (2013). Konzept und Implementierung aktivitätsbasierter Kommunikation für verteilte semantische Soziale Netzwerke. Bachelor's thesis, Universität Leipzig, Fakultät für Mathematik und Informatik, Institut für Informatik.

Snell, J., Atkins, M., Norris, W., Messina, C., Wilkinson, M., & Dolin, R. (2011). JSON Activity Streams 1.0. http://activitystrea.ms/specs/json/1.0/.

Snell, J., Atkins, M., Recordon, D., Messina, C., Keller, M., Steinberg, A., & Dolin, R. (2012). Activity Base Schema (draft). http://activitystrea.ms/specs/json/schema/activity-schema.html.

Sommerville, I. (2011). *Software Engineering* (9 ed.). Boston: Pearson Education.

Sporny, M., Inkster, T., Story, H., Harbulot, B., & Bachmann-Gmür, R. (2011). WebID 1.0 – Web Identification and Discovery. W3C Editor's Draft, W3C. http://www.w3.org/2005/Incubator/webid/spec/, Latest editor's draft: https://dvcs.w3.org/hg/WebID/raw-file/tip/spec/index-respec.html.

The-W3C-SPARQL-Working-Group (2013). SPARQL 1.1 Overview. W3C Recommendation, W3C. http://www.w3.org/TR/sparql11-overview/.

Tramp, S., Frischmuth, P., Arndt, N., Ermilov, T., & Auer, S. (2011). Weaving a Distributed, Semantic Social Network for Mobile Users. In *Proceedings of the ESWC2011*, (pp. 1–16).

Tramp, S., Frischmuth, P., Ermilov, T., & Auer, S. (2010). Weaving a Social Data Web with Semantic Pingback. In Cimiano, P. & Pinto, H. (Eds.), *Proceedings of the EKAW 2010 - Knowledge Engineering and Knowledge Management by the Masses; 11th October-15th October 2010 - Lisbon, Portugal*, volume 6317 of *Lecture Notes in Artificial Intelligence (LNAI)*, (pp. 135–149)., Berlin/Heidelberg. Springer.

Tramp, S., Frischmuth, P., Ermilov, T., Shekarpour, S., & Auer, S. (2012). An Architecture of a Distributed Semantic Social Network. *Semantic Web Journal*, *Special Issue on The Personal and Social Semantic Web*, 1–16.

Tramp, S., Frischmuth, P., & Heino, N. (2010). OntoWiki – a Semantic Data Wiki Enabling the Collaborative Creation and (Linked Data) Publication of RDF Knowledge Bases. In Corcho, O. & Voelker, J. (Eds.), *Demo Proceedings of the EKAW 2010*.

Yeung, C.-M. A., Liccardi, I., Lu, K., Seneviratne, O., & Berners-Lee, T. (2009). Decentralization: The future of online social networking. In *In W3C Workshop on the Future of Social Networking Position Papers*. http://dig.csail.mit.edu/2008/Papers/MSNWS/.

Abbildungsverzeichnis

Listings

Erklärung

Ich versichere, dass ich die vorliegende Arbeit selbständig und nur unter Verwendung der angegebenen Quellen und Hilfsmittel angefertigt habe, insbesondere sind wörtliche oder sinngemäße Zitate als solche gekennzeichnet. Mir ist bekannt, dass Zuwiderhandlung auch nachträglich zur Aberkennung des Abschlusses führen kann.

Leipzig, den 28. Juni 2013 Unterschrift