Universität Leipzig
Fakultät für Mathematik und Informatik
Institut für Informatik

Entwicklung eines mobilen Social Semantic Web Clients

Bachelorarbeit
im Studiengang Informatik

Natanael Arndt

19. Oktober 2010

Abstract

Diese Arbeit befasst sich mit der Konzeption und der Entwicklung eines mobilen Social Semantic Web Clients. Er ist auf internetfähigen Handys, Internettablets, Netbooks und weiteren Geräten mit dem Android-System einsetzbar. Der Client ermöglicht es, das Onlineprofil des Benutzers zu importieren und zu bearbeiten. Er importiert ebenfalls die darin referenzierten Profile von Bekannten von den verschiedenen verteilten Servern. Außerdem beherrscht der Client eine Authentifizierung mittels FOAF+SSL, um das Abrufen von privaten Daten zu ermöglichen. Diese Arbeit befasst sich damit, die genannte Funktionalität durch eine Middleware bereitzustellen. Dies bietet die Möglichkeit darauf aufbauende Anwendungen in Zukunft zu entwickeln. Im Rahmen dieser Arbeit wird zusätzlich eine auf diese Middleware aufbauende Benutzeroberfläche und eine Integration der Kontakte in das Android-Adressbuch entwickelt.

Schlagwörter Web ID, FOAF, Mobile, Android, Mobile Social Semantic Web, Social Network, Soziales Netzwerk, FOAF+SSL, Agent

Inhaltsverzeichnis

1 Einführung **5**
- 1.1 Motivation . . . 5
- 1.2 Ziel . . . 7
- 1.3 Aufbau . . . 7

2 Grundlagen **9**
- 2.1 Semantic Web . . . 9
- 2.2 Social Semantic Web . . . 11
- 2.3 Android-Framework . . . 12

3 Anforderungen **14**
- 3.1 Benutzeranforderungen . . . 15
  - 3.1.1 Kontaktliste des Systembenutzers abrufen . . . 15
  - 3.1.2 Informationen von Kontakten abrufen . . . 15
  - 3.1.3 Kontaktlisten der Kontakte abrufen . . . 16
  - 3.1.4 Kontakte hinzufügen . . . 16
  - 3.1.5 Synchronisation mit dem Social Semantic Web . . . 16
  - 3.1.6 Kontakte im System-Adressbuch anzeigen . . . 16
- 3.2 Funktionale Anforderungen . . . 17
  - 3.2.1 Tripelstore . . . 17
  - 3.2.2 Initialisierung der Web ID . . . 17
  - 3.2.3 Laden von Web IDs . . . 17
  - 3.2.4 Temporäres Laden von Web IDs . . . 18
  - 3.2.5 Userinterface „FOAF-Browser“ . . . 18
  - 3.2.6 Bereitstellen eines Web ID Accounttyps . . . 18
  - 3.2.7 Adressbuch Mapping . . . 18
  - 3.2.8 Synchronisationsservice . . . 19
  - 3.2.9 FOAF+SSL Authentifizierungsservice . . . 19
  - 3.2.10 Einrichten eines Clientzertifikats . . . 19
- 3.3 Nichtfunktionale Anforderungen . . . 19
  - 3.3.1 Nutzung eines autarken Tripelstores . . . 19
  - 3.3.2 Nutzung des FOAF-Vokabulars . . . 20
  - 3.3.3 Nutzung des Android-Systems . . . 20

4 Bestandsaufnahme **21**

4.1 RDF-Frameworks für mobile Endgeräte 21
4.2 Versionierung von Teilgraphen 21

5 Aufbau und Spezifikation 23
5.1 Spezifikation der Systemanforderungen 24
5.2 Semantic Web Core 25
5.2.1 Klassenübersicht des Semantic Web Cores 25
5.2.2 Abfrageschnittstellen der Klasse „TripleProvider“ 26
5.2.3 Abfrageschnittstellen der Klasse „TripleCursor“ 27
5.2.4 Spezifikation der Modellverwaltung 28
5.2.5 Spezifikation des Regelsystems 29
5.3 FOAF/WebID Provider und Browser 29
5.3.1 Klassenübersicht des FOAF/WebID Providers und Browsers . . . 29
5.3.2 Klassenübersicht des FOAF-Providers 30
5.3.3 Abfrageschnittstellen des FOAF-Providers 31
5.3.4 Abfrageschnittstellen der Klasse „PersonCursor“ 32
5.3.5 Klassenübersicht des FOAF-Browsers 33
5.3.6 Klassenübersicht der Adressbuchintegration 33
5.3.7 Beschreibung des Adressbuchvokabulars 35
5.3.8 Konfigurationsassistent 36
5.3.9 Auflösung von URIs durch die Klasse „NameHelper“ 36

6 Implementierung 37
6.1 Konstante Werte 37
6.2 Mobile Semantic Web Middleware 37
6.2.1 Implementierung der Modellverwaltung 38
6.2.2 Implementierung des Regelsystems 39
6.3 Mobile Social Semantic Web Middleware 40
6.3.1 Implementierung des FOAF-Providers 40
6.3.2 Implementierung des FOAF-Browsers 40
6.3.3 Implementierung der Adressbuchintegration 42
6.3.4 Auflösung von URIs durch die Klasse „NameHelper“ 45

7 Zusammenfassung und Ausblick 46
7.1 Verbesserungsmöglichkeiten des Systems 46
7.2 Möglichkeiten aufbauender Applikationen 48

8 Quellcode der Software 49

Literaturverzeichnis 50

Abbildungsverzeichnis 52

Listings 53

1 Einführung

1.1 Motivation

Soziale Netzwerke spielen eine immer größer werdende Rolle in unserem täglichen Leben und die Einrichtung und Pflege unserer Accounts und Daten in diesen Netzwerken wird immer aufwendiger. Die meisten sozialen Netzwerke sind in sich geschlossen. Nutzer können ihre Daten häufig nicht von einem Netzwerk zu einem anderen kopieren. Freundschaften zwischen Netzwerken sind nicht möglich und man muss sich in jedem Netzwerk ein neues Profil anlegen, es gilt *one-profile-per-network*, wie es in Abbildung 1.1 schematisch dargestellt wird. Die grüne Person benötigt drei Profile um mit allen Freunden in Kontakt zu stehen.

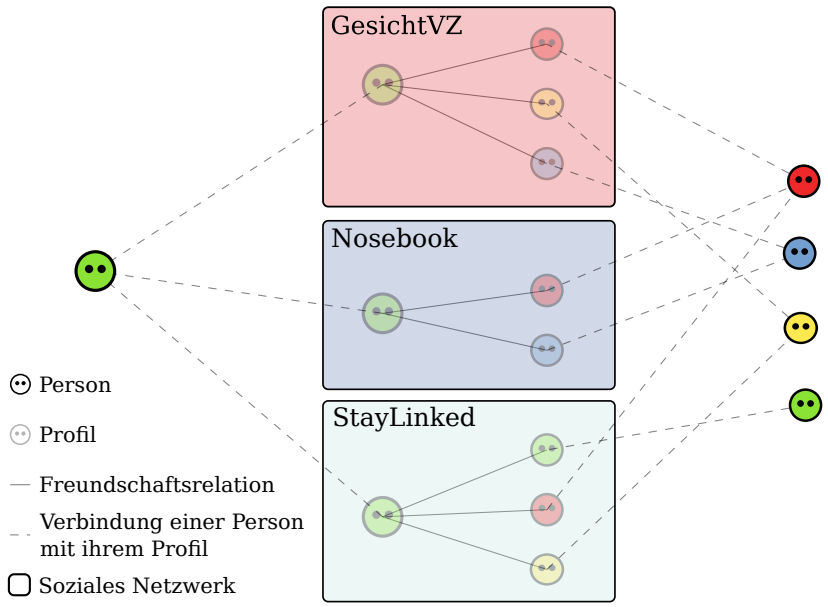

Abbildung 1.1: Heutzutage muss sich jede Person in jedem sozialen Netzwerk ein Profil anlegen

Das uns bekannte, von Tim Berners-Lee beschriebene, World Wide Web (WWW) kennt diese Art künstlicher Abtrennung einzelner Teilnetze nicht:

> „The idea of a boundless information world in which all items have a reference by which they can be retrieved“ Berners-Lee u. a. [1994]

Das *Social Semantic Web* (siehe Abschnitt 2.2) ist die Idee eines dezentralisierten und serviceunabhängigen Sozialen Netzwerkes, welches auf dem WWW und Technologien des Semantic Webs (siehe Abschnitt 2.1) aufbaut. Jeder Person wird dabei eine Web ID (mehr in Abschnitt 2.2) zugeordnet, welche die Person eindeutig identifiziert (*one-profile-per-person*) und die Möglichkeit bietet persönliche Beziehungen verschiedener Personen untereinander darzustellen. Diese Netzstruktur wird schematisch in Abbildung 1.2 dargestellt. Jeder Person benötigt nur ein Profil um am Social Semantic Web teilzunehmen.

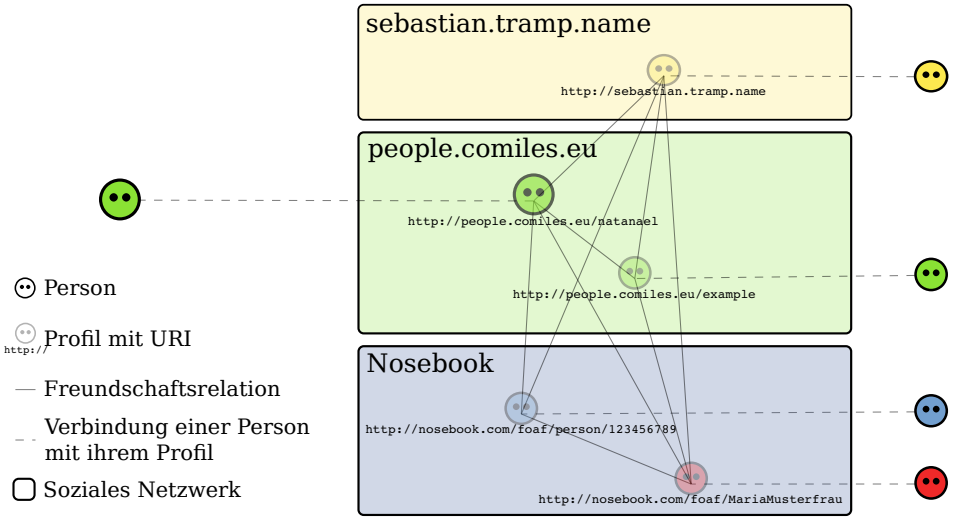

Abbildung 1.2: Im Social Semantic Web sind Beziehungen zwischen Profilen auf unterschiedlichen Servern möglich

Mobile Kommunikation gewinnt immer mehr an Bedeutung. Die meisten der großen herkömmlichen sozialen Netzwerke bieten auch bereits Möglichkeiten zur Integration der Kontaktdaten in mobile Plattformen an. Jedoch existiert bisher noch keine derartige Möglichkeit für das Social Semantic Web. Die Idee des *Social Semantic Web Clients* ist es daher, Teilnehmern dieses Social Semantic Webs eine Möglichkeit zu geben, mit mobilen Geräten in ihm zu navigieren und es entsprechend zu manipulieren (z. B. neue Bekanntschaften dem eigenen Profil hinzuzufügen).

Über die grundlegenden Funktionen sozialer Netze hinaus wird auch die Technik *FOAF+SSL* (mehr in Abschnitt 2.2) implementiert. Sie bietet die Möglichkeit, sich gegenüber einem Webserver zu authentifizieren und dadurch zugangsbeschränkte Zusatzinformationen einer Web ID zu lesen.

1.2 Ziel

Ziel der Arbeit ist der Entwurf und die Implementierung eines mobilen Social Semantic Web Clients. Zu diesem Zweck wird eine Middleware entwickelt und beispielhaft für die Android-Plattform[1] implementiert (vergleiche dazu „Anwendungs-Framework-Ebene“ in Abbildung 5.1). Diese Middleware agiert dabei einerseits als Client (*Agent*) des Semantic Webs und stellt andererseits die daraus gewonnenen Daten auf der mobilen Plattform zu Verfügung. Auf diese Middleware können verschiedene Programme einheitlich zugreifen. Die Daten können von den Programmen weiterverarbeitet, dem Nutzer präsentiert und in andere Anwendungsfälle integriert werden. Ebenfalls werden zwei derartige Anwendungen entwickelt, ein FOAF-Browser und die Adressbuchsynchronisation. Die Architektur des Social Semantic Web Clients wird modular konzipiert, um eine einfache Erweiterbarkeit des Funktionsumfangs zu ermöglichen. (Möglichkeiten zur Weiterentwicklung befinden sich in Abschnitt 7.)

1.3 Aufbau

Nach der „Einführung“ (siehe Abschnitt 1) wird im Kapitel „Grundlagen“ (siehe Abschnitt 2) das Konzept eines Social Semantic Webs erläutert. Dabei wird auf bestehende Semantic Web-Technologien eingegangen und das Social Semantic Web Vokabular FOAF mit den darauf aufbauenden Konzepten Web ID und FOAF+SSL vorgestellt. Außerdem werden zwei wichtige Konzepte der Android-Plattform erklärt.

Im Kapitel „Anforderungen“ (siehe Abschnitt 3) wird darstellt, welche Funktionen und Eigenschaften das Endprodukt der Arbeit aufweisen soll. Zuerst werden „Benutzeranforderungen“ (siehe Abschnitt 3.1) aufgestellt und aus diesen die nötigen funktionalen (siehe Abschnitt 3.2) und nichtfunktionalen (siehe Abschnitt 3.3) Anforderungen extrahieren.

Im Kapitel „Bestandsaufnahme“ (siehe Abschnitt 4) wird ein Blick auf den aktuellen Stand der Technik und Forschung geworfen und erörtert, welche Projekte und Programme zur Lösung des Problems genutzt werden können.

Das Kapitel „Aufbau und Spezifikation“ (siehe Abschnitt 5) beschreibt die nötigen Systemvoraussetzungen zum Betrieb der entstandenen Software. Es werden die Programmstruktur, anhand von Klassendiagrammen, beschrieben und die definierten Schnittstellen, zur Weiterverwendung der Middleware, dokumentiert.

[1] Android: `http://www.android.com/`, `http://developer.android.com/`

Das Kapitel „Implementierung“ (siehe Abschnitt 6) fasst die Entwicklungsarbeit der Software zusammen und beschreibt die Funktionsweise einzelner Teilkomponenten näher.

Gegen Ende wird im Kapitel „Zusammenfassung und Ausblick“ (siehe Abschnitt 7) die getane Arbeit zusammengefasst und es werden einige Anregungen zur Nutzung der in dieser Arbeit entwickelten Middleware, durch aufbauende Software, gegeben. Da ein Programm nie fertig sein kann werden auch Möglichkeiten der Ergänzung und Verbesserung der Funktionalität der entstandenen Software aufgezeigt.

Woher der Quellcode der Software bezogen werden kann steht in Abschnitt 8, gefolgt von einigen Verzeichnissen und einem Extended Abstract zu diesem Projekt im Anhang (Arndt [2010]).

2 Grundlagen

In diesem Kapitel werden einige für diese Arbeit grundlegende Techniken des Semantic Webs, Social Semantic Webs und der verwendeten Android-Plattform vorgestellt.

2.1 Semantic Web

Das Semantic Web, wie es Berners-Lee u. a. (Berners-Lee u. a. [2001]) beschreiben, ist wie das World-Wide Web (WWW) eine weltumspannende Datenbank aus verteilten Dokumenten. Diese Dokumente können aber im Gegensatz zum herkömmlichen WWW durch Maschinen oder Agenten interpretiert werden, da die Daten strukturiert vorliegen. Ein solches Dokument beinhaltet eine *Wissensbasis*, sie fasst meist Informationen über ein bestimmtes Thema oder ein Objekt zusammen. Die Daten werden mit Hilfe des Resource Description Frameworks (RDF, Lassila u. Swick [1999]) ausgedrückt.

RDF – Resource Description Framework

Im Resource Description Framework werden Ressourcen beschrieben. Eine Ressource kann ein Web-Dokument, ein Abschnitt eines Dokuments oder irgend ein anderes Ding sein, das durch einen *Universal Resource Identifier* (URI, Berners-Lee u. a. [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 zu anderen Ressourcen ausdrücken.

Die Eigenschaften werden durch Aussagen (auch *Tripel* oder *Statements*) formuliert. Eine Aussage besteht aus drei Bestandteilen, einer Ressource, deren Eigenschaft und dem entsprechenden Wert dieser Eigenschaft. Diese drei Teile werden auch Subjekt, Prädikat und Objekt genannt. Das Objekt kann eine Ressource mit URI oder ein Literal sein.

Vokabular Ein RDF Vokabular ist eine Sammlung von Ressourcen und Eigenschaften. Sie dienen dazu häufig auftretende Strukturen zu vereinheitlichen. Meist sind diese in einem gemeinsamen Namensraum definiert. Der gleich bleibende Teil des URI wird in der Regel durch ein *Präfix* (engl. *Prefix*) abgekürzt. Die in dieser Arbeit verwendeten Präfixe sind in Listing 2.1 definiert.

```
1 @prefix rdf: <http://www.w3.org/1999/02/22-rdf-syntax-ns#>.
2 @prefix rdfs: <http://www.w3.org/2000/01/rdf-schema#>.
3 @prefix sioc: <http://rdfs.org/sioc/ns#>.
4 @prefix foaf: <http://xmlns.com/foaf/0.1/>.
5 @prefix android: <http://ns.aksw.org/Android/>.
6 @prefix acontacts: <http://ns.aksw.org/Android/ContactsContract.
    CommonDataKinds.>.
7 @prefix name: <http://ns.aksw.org/Android/ContactsContract.
    CommonDataKinds.StructuredName.>.
8 @prefix phone: <http://ns.aksw.org/Android/ContactsContract.
    CommonDataKinds.Phone.>.
9 @prefix email: <http://ns.aksw.org/Android/ContactsContract.
    CommonDataKinds.Email.>.
```

Listing 2.1: In dieser Arbeit verwendete RDF-Präfixe

Darstellung als Graph Wie in Abbildung 2.1 schematisch visualisiert, kann eine RDF-Wissensbasis auch in Form eines Graphen dargestellt werden. Eine Ressource wird durch einen Knoten repräsentiert (im Bild die gelben Kreise). Ressourcen werden durch Eigenschaften in Form von gerichteten und typisierten Kanten verbunden (im Bild durch unterschiedliche gefärbte Pfeile dargestellt). Literale können nur an den Enden des Graphen als Blätter auftreten (der grüne Kreis im Bild). Auch Ressourcen, welche nur in einer Aussage auftreten, können als Blätter erscheinen (der rechte gelbe Kreis im Bild).

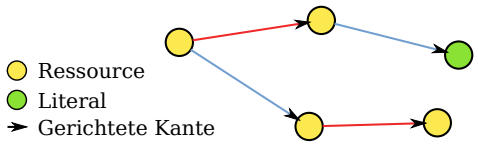

Abbildung 2.1: Darstellung eines RDF-Graphen mit Ressourcen, gerichteten und typisierten Kanten und einem Literal

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.

Durch URIs können Ressourcen weltweit eindeutig identifiziert werden (vergleiche Punkt 1), ohne Namenskonflikte zu befürchten, solange URIs vergeben werden, über deren Namensraum man selbst Herr ist. Indem URIs mit dem HTTP-Schema verwendet werden, können sie über einen weit verbreiteten Standard angefragt werden (vergleiche Punkt 2). Auf eine Anfrage dieser URI sollte mit strukturierten, für Maschinen verständlichen Daten geantwortet werden. Nach Möglichkeit in einem weit verbreiteten Format (vergleiche 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 (vergleiche 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 Social Semantic Web

Das *Social Semantic Web* ist die Idee eines dezentralisierten und serviceunabhängigen sozialen Netzwerkes, welches auf den Technologien des Semantic Webs aufbaut. Personen werden als Ressourcen in RDF-Dokumenten beschrieben. Beziehungen zwischen Personen werden als Eigenschaften modelliert. Betrachtet man dieses Netzwerk als Graph erhält man einen Freundschafts- oder Beziehungsgraphen. Die URI einer Person wird auch Web ID[1] (siehe Abschnitt 2.2) genannt. Werden diese Web IDs gemäß der Linked-Data-Regeln (siehe Abschnitt 2.1) veröffentlicht, kann man darin wie in einem herkömmlichen sozialen Netzwerk navigieren. Es entsteht ein verteiltes Social Semantic Web.

FOAF – Das Friend of a Friend Vokabular

Das Friend of a Friend (FOAF) Projekt entwickelt ein RDF Vokabular (Brickley u. Miller [2004]). Es ermöglicht auf einheitliche Weise Personen durch deren Namen, JabberID, Weblog, u. Ä. zu beschreiben und deren persönliche Beziehungen untereinander

[1] Web ID: `http://esw.w3.org/WebID`

darzustellen. Im FOAF-Vokabular ist die einfachste Beziehung zwischen Personen formuliert, `foaf:knows`. Die Beziehung `foaf:knows` wird durch das Relations-Vokabular (`rel:`, Davis u. Jr [2010]) mit Verwandtschafts-, verschieden abgestuften Freundschafts- und vielen anderen Beziehungen erweitert.

Web ID – Das Onlineprofil

> „The WebID specification is designed to help alleviate the difficultly that remembering different logins, passwords and settings for websites has created. It is also designed to provide a universal and extensible mechanism to express public and private information about yourself.“ Inkster u. a. [2010]

Eine Web ID ist in erster Linie ein URI zur Identifikation einer Person. Wird eine Web ID abgefragt erhält man ein RDF-Dokument mit einer Beschreibung dieser Person. Diese Beschreibung ist meist hauptsächlich im FOAF-Vokabular (siehe Abschnitt 2.2) formuliert. Ein weiterer Aspekt der Web ID ist die Möglichkeit diese zur Identifikation gegenüber Internetdiensten zu nutzen. Dieser Aspekt wird im Abschnitt 2.2 besprochen. Das Konzept der Web ID ist in weiten Teilen vergleichbar mit Profilen in herkömmlichen sozialen Netzwerken.

FOAF+SSL – Sicherheit für das Social Web

FOAF+SSL beschreiben Story u.a. (Story u. a. [2009]) als ein offenes Verfahren zur sicheren Identifikation einer Person durch deren Web ID in einem „one-click sign-on“-Verfahren. Dabei ist der Identifikationsvorgang nicht auf eine zentrale Identifikations-Instanz angewiesen, sondern jeder Nutzer kann an einer anderen Stelle registriert sein. Es ist in dieser Hinsicht vergleichbar mit der bekannten verteilten OpenID-Technologie, welche ebenfalls einen URI zur Identifikation verwendet.

2.3 Android-Framework

Diese Software wird für die mobile Plattform Android[2] entwickelt. Für die Softwareentwicklung wird ein Software Development Kit (SDK)[3] ausgeliefert, welches es ermöglicht

[2] Android: `http://developer.android.com/`
[3] Android SDK: `http://developer.android.com/sdk/index.html`

das Android-Application-Framework[4] zu nutzen. Zwei für diese Arbeit sehr wichtige Konzept des Application-Frameworks sind Content-Provider und Intents.

Content-Provider-Schnittstelle

Auf der Android-Plattform läuft jede Anwendung in einem eigenen Prozess. Ein Content-Provider[5] bietet die Möglichkeit Prozessübergreifend Daten über eine flexible Schnittstelle auszutauschen, was für eine Middleware essenziell ist. Ein Content-Provider wird über dreiteilige URIs angefragt, welche sich folgendermaßen zusammensetzen:

Schema Das Verwendete Schema zur Anfrage eines Content-Providers ist `content://`.

Authority der volle Java-Pfad zu der Klasse, welche die Klasse `ContentProvider` erweitert. In dieser Arbeit sind es `org.aksw.msw.tripleprovider`, `org.aksw.mssw.foafprovider` und `org.aksw.mssw.contactprovider`.

Pfad ein frei wählbarer Pfad. Die in dieser Arbeit verwendeten Pfade werden in Abschnitt 5 spezifiziert.

Anfragen an Content-Provider werden mit `Cursor`-Objekten beantwortet. Die Mobile Semantic Web Middleware und die Mobile Social Semantic Web Middleware sind über eine Content-Provider-Schnittstelle von anderen Programmen aus abfragbar.

Intents

Eine Weitere Möglichkeit der Kommunikation zwischen Prozessen ist es eine Vorhaben (engl. *Intent*[6]) zu formulieren. Das Android-System sucht nach einem passenden Programm, welches dieses Vorhaben erfüllen kann. Mit einem Intent kann ein URI (Intent-Data) und Zusätzliche Daten („Extra“) mitgesendet werden

[4] What is Android?: `http://developer.android.com/guide/basics/what-is-android.html`
[5] Content-Provider: `http://developer.android.com/guide/topics/providers/content-providers.html`
[6] Intents: `http://developer.android.com/guide/topics/intents/intents-filters.html`

3 Anforderungen

In diesem Kapitel werden die Anforderungen an das zu entwickelnde System formuliert. Dabei werden die wichtigsten Grundfunktionen erörtert, welche durch die Middleware bereitgestellt werden müssen. Durch die Kombination dieser Grundfunktionen können später auf diese Middleware aufbauende Anwendungen dem Benutzer komplexere Funktionen anbieten. Diese Arbeit wird sich daher auch nicht mit allen Eventualitäten der Anwendungsvielfalt befassen, da die Middleware in einer späteren Version erweitert werden kann.

Die gesamte Infrastruktur wird, wie in Abbildung 3.1 schematisch dargestellt, im Kontext eines bereits bestehenden sozialen Netzwerkes eingesetzt. Die Middleware soll anderen Anwendungen eine Grundlage zur Navigation im Social Semantic Web bieten.

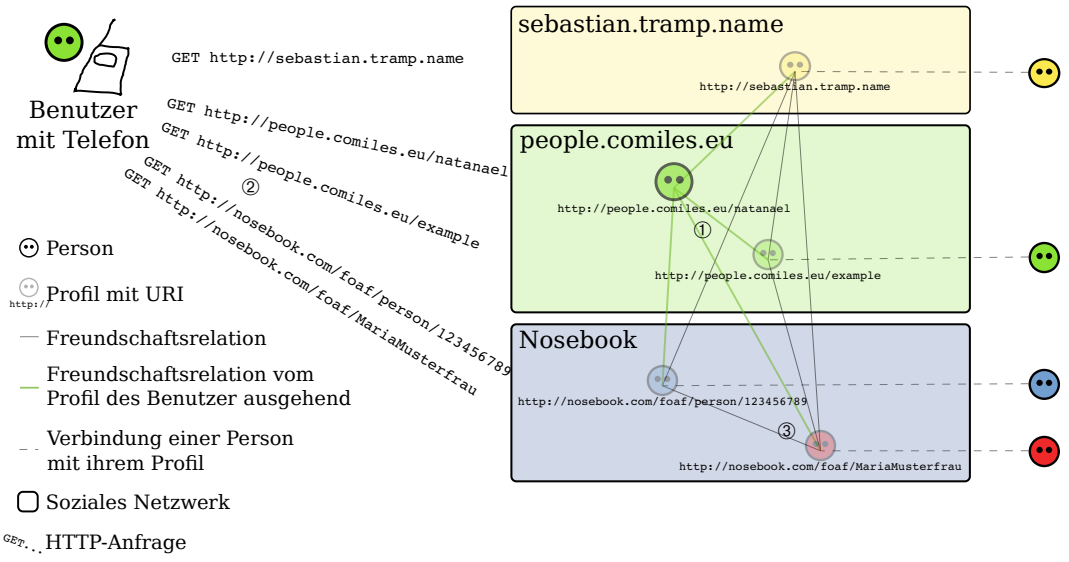

Abbildung 3.1: Abfrage von Profilen (2) und Kontaktlisten (1 und 3) aus dem Social Semantic Web

3.1 Benutzeranforderungen

> „Die Benutzeranforderungen für ein System sollten die funktionalen und nichtfunktionalen Anforderungen so beschreiben, dass sie für Systembenutzer verständlich sind, die kein detailliertes technisches Wissen haben. Sie sollten nur das externe Verhalten des Systems festlegen und Charakteristika des Systementwurfs so weit wie möglich vermeiden.“ [Sommerville, 2007, 159]

Da die hier entwickelte Anwendung eine Middleware ist, sind die Systembenutzer hauptsächlich Anwendungsprogrammierer. Es wird aber ebenfalls eine Beispielapplikation entwickelt, die von Endanwendern bedient werden kann. Im Folgenden wird sowohl für den Endanwender als auch für den Anwendungsprogrammierer der Begriff „Systembenutzer“ oder auch „Benutzer“ verwendet, da eine Unterscheidung nur künstlich möglich wäre und die Grenzen in einigen Fällen verschwimmen.

3.1.1 Kontaktliste des Systembenutzers abrufen

In der Web ID des Systembenutzers sind Beziehungen zu anderen Web IDs, und damit den dadurch referenzierten Personen (Kontakten), definiert (siehe Abbildung 3.1: Punkt 1). Diese Beziehungen können unterschiedlicher Art sein, um auszudrücken wie die Personen zueinander stehen, zum Beispiel Freunde sind oder miteinander arbeiten. Es soll eine Liste der Kontakte erstellt werden, welche mit der Web ID des Systembenutzers in Verbindung stehen. Dabei soll nach Art der Beziehungen unterschieden werden.

Auch in anderen Web IDs, die in das System geladen wurden, können Beziehungen, an denen die Web ID des Benutzers beteiligt ist definiert sein. Es stellt sich dabei die Frage, ob diese Beziehungen gesondert behandelt werden sollen oder ob die Herkunft der Relationen unwichtig ist.

3.1.2 Informationen von Kontakten abrufen

Da das Social Semantic Web verteilt organisiert ist werden die Web IDs der Kontakte von verschiedenen Webservern nach den Linked Data-Regeln (vergleiche Abschnitt 2.1 und Abbildung 3.1: Punkt 2) importiert. Aus Gründen der Wahrung der Privatsphäre sind Teile der in einer Web ID enthaltenen Informationen nicht allgemein zugänglich. Das private Profil wird erst nach einer Authentifizierung freigegeben. Hierzu soll FOAF+SSL eingesetzt werden. Es bedarf also einer Identifikation des Social Semantic Web Clients gegenüber dem Webserver, wobei sich der Client als der Benutzer selbst bzw. im Auftrag dessen handelnd ausgibt.

3.1.3 Kontaktlisten der Kontakte abrufen

Da sich diese Arbeit mit einem Netz von Kontakten befasst, ist nach der ersten Kante (in der Darstellung als Graph, Abschnitt 2.1, siehe auch Abbildung 3.1: Punkt 3) noch lange nicht Schluss. Es muss also eine Möglichkeit bestehen, auch die Kontakte von Kontakten („Friend of a Friend“) anzuzeigen. Der Systembenutzer kann in seiner eigenen Kontaktliste navigieren und eine Web ID auswählen, die mit ihm in Verbindung steht. Er erhält, eventuell zusammen mit den anderen Informationen des Kontakts, eine Liste der Web IDs, die mit dieser Person in Verbindung stehen.

3.1.4 Kontakte hinzufügen

Über eine Bearbeiten-Funktion wird dem Benutzer die Möglichkeit gegeben zu der eigenen Web ID Verbindungen zu neuen Web IDs hinzuzufügen. Dabei kann er sowohl eine beliebige bereits im System angezeigte Web ID wählen oder mit der Tastatur eine neue Web ID hinzufügen.

3.1.5 Synchronisation mit dem Social Semantic Web

Der lokale Stand des Social Semantic Webs kann sehr schnell veralten, da die einzelnen Web IDs von unterschiedlichen Personen zur gleichen Zeit bearbeitet werden können. Deshalb müssen lokale Änderungen der Web ID des Systembenutzers mit der im Internet verfügbaren Version auf dem Webserver synchron gehalten werden. Ebenfalls müssen die lokal vorgehaltenen Web IDs der Kontakte auf Aktualität überprüft werden.

3.1.6 Kontakte im System-Adressbuch anzeigen

Die in der Web ID des Benutzers referenzierten Kontakte sollen im Systemadressbuch angezeigt werden. Das Systemadressbuch enthält in den meisten Fällen bereits Einträge. Unter Umständen ist also auch eine Verknüpfung der Web IDs mit den Einträgen des Adressbuchs vorzunehmen. Dabei sollen so viele Eigenschaften der Kontakte wie möglich in das Systemadressbuch integriert werden.

Automatisch aktualisierendes Adressbuch Implizit ergibt sich daraus ein sich automatisch aktualisierendes Adressbuch. Da jeder Benutzer seine eigene Web ID pflegt,

erhält jeder Nutzer auf seinem Gerät automatisch ein aktuelles Adressbuch mit den Daten seiner Kontakte.

3.2 Funktionale Anforderungen

Aus den oben formulierten Benutzeranforderungen ergeben sich funktionale Systemanforderungen, welche dazu dienen das zu entwickelnde System zu entwerfen.

3.2.1 Tripelstore

Das Social Semantic Web setzt sich aus Tripeln zusammen. Diese können in verschiedenen Formaten repräsentiert werden. Eine ständige Serialisierung und Deserialisierung der verwendeten Daten wäre allerdings zu aufwendig. Daher muss ein minimaler Tripelstore (oder auch RDF-Store) zur Speicherung der Web IDs und ihrer Eigenschaften eingesetzt werden.

3.2.2 Initialisierung der Web ID

Um die Kontaktliste des Systembenutzers abrufen zu können (wie es in Abschnitt 3.1.1 gefordert wird) muss die Web ID des Systembenutzers in den Tripelstore geladen werden. Dies könnte man so realisieren, dass der Systembenutzer beim erstmaligen Öffnen des FOAF-Browsers (siehe Abschnitt 3.2.5) nach der Installation mit einem Einrichtungsassistenten konfrontiert wird. Der Nutzer gibt den URI seiner Web ID ein. Anschließend werden seine Profildaten und die Profildaten seiner Kontakte geladen. Diese Funktion sollte auch zu einem späteren Zeitpunkt manuell gestartet werden können, um eine fehlerhafte lokale Kopie der Web ID zu überschreiben. Diese Fehler können zum Beispiel durch Fehler im Programm hervorgerufen werden und würden nach einem Update der Software nicht mehr auftreten. Diese Funktion kann sich mit dem „Synchronisationsservice“ (wie er in Abschnitt 3.2.8 beschrieben wird) überschneiden.

3.2.3 Laden von Web IDs

Das Laden von Web IDs ist eine Generalisierung der Anforderung „Initialisierung der Web ID“ (siehe Abschnitt 3.2.2). Es werden beliebige Web IDs nach den Linked Data-Regeln (vergleiche Abschnitt 2.1) in den Tripelstore (siehe Abschnitt 3.2.1) geladen.

Diese Daten sind dann auch ohne Internetverbindung erreichbar. In dem Dokument, welches von dem Webserver bezogen wird, können mehrere `foaf:Persons` beschrieben sein. Es sollte durch einen Filter bzw. eine SPARQL-Anfrage verhindert werden, dass Aussagen, die nicht die gewünschte Web ID als Subjekt besitzen, in den Tripelstore importiert werden.

3.2.4 Temporäres Laden von Web IDs

Um die Kontakte der Kontakte (siehe Abschnitt 3.1.3) anzuzeigen müssen deren Web IDs geladen werden. Allerdings ist keine permanente Speicherung dieser Daten notwendig und sinnvoll. Es werden also Web IDs, wie in Abschnitt 3.2.3 „Laden von Web IDs“ beschrieben, geladen und nach einer gewissen Zeit wieder aus dem Tripelstore gelöscht.

3.2.5 Userinterface „FOAF-Browser“

Der FOAF-Browser wird als Beispielapplikation implementiert. Er ist ein Benutzerinterface, welches zur Demonstration der Funktionalität der Middleware dient. Diese Applikation bietet Funktionen an, um neue Kontakte zur Web ID des Benutzers hinzufügen zu können. Außerdem bietet er eine Möglichkeit an, die Web ID und die Kontakte des Benutzers zu betrachten. Über die Liste der Kontakte kann man zur Ansicht deren Web IDs gelangen.

3.2.6 Bereitstellen eines Web ID Accounttyps

Um die Kontakte im Systemadressbuch anzeigen zu können (wie in Abschnitt 3.1.6 gefordert) muss auf Android-Systemen ein neuer Accounttyp für Kontakte erstellt werden. Dazu muss ein `AccountAuthenticator` und ein `SyncAdapter` implementiert werden.

3.2.7 Adressbuch Mapping

Da in einer Web ID sehr vielfältige Möglichkeiten bestehen, um Daten zu formulieren kann eine Web ID nicht eins zu eins in das Systemadressbuch integriert werden. Zu diesem Zweck gibt es ein Mapping, das der Benutzer selbst bearbeiten kann. In diesem wird festgelegt welche Eigenschaften welchen Adressbuchfeldern zugeordnet werden. Dabei sollten auch Pfade (zum Beispiel über Blank Nodes oder andere Ressourcen) berücksichtigt

werden. Dieses Mapping sollte in RDF formuliert hinterlegt sein. Möglicherweise könnte das Mapping auch im Internet hinterlegt sein, um es automatisch zu aktualisieren.

3.2.8 Synchronisationsservice

Einerseits sollten auf dem Social Semantic Web Client geänderte Eigenschaften an den entsprechenden Webserver per Semantic-Pingback zurückgemeldet werden, andererseits sollte aber auch eine Übermittlung der getätigten Änderungen an nicht Semantic-Pingback fähige Webserver per SPARQL/Update möglich sein. Diese Funktion sollte konfigurierbar automatisch gestartet werden, um im Hintergrund die Web IDs auf dem neusten Stand zu halten. Sie sollte aber auch manuell gestartet werden können.

3.2.9 FOAF+SSL Authentifizierungsservice

Implementierung das FOAF+SSL-Authentifizierungsprotokolls, um geschützte Bereiche einer Web ID laden zu können.

3.2.10 Einrichten eines Clientzertifikats

Der Benutzer kann ein Clientzertifikat für die Nutzung mit FOAF+SSL von der SD-Karte importieren oder ein neues direkt auf dem mobilen System generieren lassen.

3.3 Nichtfunktionale Anforderungen

Folgende nichtfunktionale Anforderungen sollten bei dem Entwurf das Systems berücksichtigt werden.

3.3.1 Nutzung eines autarken Tripelstores

Der eingesetzte Tripelstore (beschrieben in Abschnitt 3.2.1) sollte separat vom restlichen System eingerichtet werden und als `ContentProvider` zugänglich sein, um ihn auch von externen Programmen aus nutzen zu können.

3.3.2 Nutzung des FOAF-Vokabulars

Das FOAF-Vokabular sollte zu eine Großen Teil interpretiert werden, da es für die Beschreibung von Personen und sozialen Beziehungen das bekannteste Vokabular ist.

3.3.3 Nutzung des Android-Systems

Android ist ein sehr weit verbreitetes mobiles Betriebssystem (vergleiche hierzu auch Abschnitt 2.3). Es bietet eine umfangreiche Dokumentation der Programmierschnittstelle[1] und basiert auf dem freien Linux Kernel. Auf der Android-Plattform kommt hauptsächlich Java-Bytecode in dem speziellen Dalvik-Format zur Ausführung[2]. Außerdem verwenden die meisten Mitarbeiter der Arbeitsgruppe (Agile Knowledge Engineering and Semantic Web, AKSW[3]) in der diese Arbeit entsteht Android, wodurch ein bestehende Nutzerschaft geboten ist.

[1] Android Reference: `http://developer.android.com/reference/packages.html`
[2] Android Runtime: `http://developer.android.com/guide/basics/what-is-android.html#runtime`
[3] AKSW: `http://aksw.org/`

4 Bestandsaufnahme

In diesem Abschnitt wird ein Blick auf den aktuellen Stand der Technik in dem Bereich das mobilen Social Semantic Webs geworfen. Es wird ein RDF-Framework zum Einsatz auf mobilen Endgeräten und eine Technik zum Herunterladen von Ausschnitten einer RDF-Wissensbasen von einem Webserver vorgestellt.

4.1 RDF-Frameworks für mobile Endgeräte

Für das Parsen und Laden der unterschiedlichen Serialisierungsformate (RDF/XML, Turtle, ...), die persistente Speicherung und das Abfragen der RDF-Wissensbasen ist ein RDF-Framework vorteilhaft. Das wohl bekannteste RDF-Framework für Java ist das „Jena Semantic Web Framework“. Es bietet die Vorteile, dass es Open-Source-Software ist, weit Verbreitet und damit gut Dokumentiert ist. Im Androjena-Projekt[1] wird das Jena Framework auf die Android-Plattform portiert. Zu Beginn dieser Arbeit war noch keine Möglichkeit der Ausführung von SPARQL vorhanden, dies hat sich aber mittlerweile geändert (vergleiche Abschnitt 7.1). Außerdem besitzt es ein sehr flexibles Regelsystem, welches für die Adressbuchintegration genutzt wird (siehe Abschnitt 5.2.5).

4.2 Versionierung von Teilgraphen

In „Replication and Versioning of Partial RDF Graphs“ beschreibt Schandl (Schandl [2010]) ein Verfahren zum Kopieren eines Teilgraphen aus einem größeren Graphen. Dieses Verfahren ist speziell für den Einsatz auf mobilen Geräten vorgesehen, da sie oft nicht über die nötigen Ressourcen verfügen, um einen großen vollständigen Graphen zu handhaben. Dabei werden nicht nur Rechen- und Speicherressourcen auf dem Client eingespart, sondern außerdem zu übertragendes Datenvolumen, was ebenfalls dem mobilen Einsatz zugute kommt.

[1] androjena – Jena Android porting: `http://code.google.com/p/androjena/`

Schandl zeigt die Einsatzmöglichkeit für die Replikation aus der Linked Data Cloud auf, was für die hier behandelte Problemstellung passend ist (vergleiche Anforderung „Synchronisation mit dem Social Semantic Web“ in Abschnitt 3.1.5). Da diese Technik noch sehr jung ist, kann man sich momentan noch nicht auf eine breite Verfügbarkeit von Repositories verlassen. Dies wird sich hoffentlich in den nächsten Jahren vor allem durch den stark wachsenden Markt mobiler Kommunikationsgeräte ändern.

Eine abgewandelte Form dieser Technik könnte aber für die Adressbuchsynchronisation (vergleiche Abschnitt 3.1.6) zum Einsatz kommen, da nur eine Teilmenge der auf der mobilen Plattform vorhandenen Daten in das Adressbuch übertragen werden. Der Contact-Provider würde dabei als Repository auftreten, das Android-Contact-Contract-System als Client. Die Bitmaps könnten in einem der frei belegbaren `SYNC<1-4>` Felder der Kontakte abgelegt werden (vergleiche Abschnitt 5.3.7). Allerdings würde im Gegensatz zu Schandls Vorschlag nicht der Client den Teilgraphen anfordern, sondern eine vorgelagerte Ebene würde dies übernehmen und die Daten in das Adressbuch einfügen.

5 Aufbau und Spezifikation

Diese Anwendung läuft auf dem Android-Betriebssystem für mobile Kommunikationsgeräte. Sie stellt eine Middleware für andere Anwendungen zur Navigation im Semantic Web und speziell im Social Semantic Web zu Verfügung. Abbildung 5.1 stellt die Integration der Middleware und der Benutzeroberfläche in die bestehende Android-Architektur[1] dar.

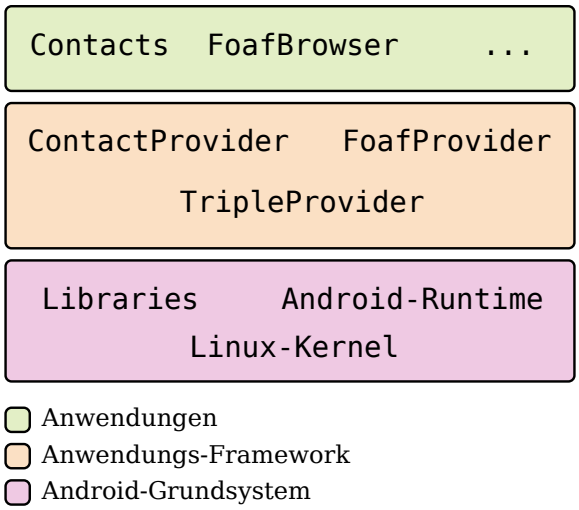

Abbildung 5.1: Integration des Social Semantic Web Clients in das bestehende Android-System

Die Android-Architektur ist in fünf Bereiche unterteilt, eine Anwendungs-Ebene, eine Anwendungs-Framework-Ebene, eine Sammlung von Bibliotheken (`Libraries`), die Android-Laufzeitumgebung (`Android Runtime`) unter Anderem mit der „Dalvik Virtual Machine“ und dem Linux-Kernel [Meier, 2010, 13 f]. Die untersten drei Bereiche wurden hier in eine Ebene „Android-Grundsystem“ zusammengefasst. Für Anwendungsentwickler ist es möglich Programme auf der Anwendungs- und Anwendungs-Framework-Ebene auszuführen. Auf der Anwendungs-Framework-Ebene wird die Middleware zur Bereitstellung von Daten und Funktionen, zur Nutzung durch die Anwendungs-Ebene,

[1] Android Architecture: `http://developer.android.com/guide/basics/what-is-android.html#os_architecture`

ausgeführt. Der `TripleProvider` stellt grundlegende Semantic Web Funktionalität zur Verfügung. Der `ContactProvider` und der `FoafProvider` stellen spezielle soziale Funktionen auf den Daten des `TripleProvider`s zur Verfügung. Auf der Anwendungs-Ebene laufen für den Benutzer unmittelbar sichtbare Programme (*Activities*). Die `Contacts`-Activity wird mit den meisten Android-Versionen ausgeliefert und stellt unter anderem das Systemadressbuch dar. Der `FoafBrowser` ist eine im Rahmen dieses Projekts entwickelte Benutzeroberfläche, um die sozialen Funktionen des `FoafProvider`s bedienen zu können.

Zur Trennung der grundlegenden Semantic Web Funktionalität von der sozialen Komponente wurde das Gesamtprojekt in zwei Projekte unterteilt. Eine Mobile Semantic Web Middleware („Semantic Web Core“, Abschnitt 5.2) mit dem `TripleProvider` und eine Mobile Social Semantic Web Middlewares („FOAF/WebID Provider und Browser“, Abschnitt 5.3) `FoafBrowser`, `FoafProvider` und `ContactProvider`.

Anmerkung Alle Klassendiagramme in diesem Dokument wurden nach dem UML-Standard (OMG [2005]) gezeichnet.

5.1 Spezifikation der Systemanforderungen

Die Software soll auf der frühestmöglichen Android-Version funktionieren, aber auch zukünftige Versionen des Systems unterstützen. Die momentan aktuelle Version ist 2.2 (API Level 8).

FOAF/WebID Provider und Browser Mit Android 2.0 (API Level 5) wurde die `ContactsContract` Klasse eingeführt. Hierdurch werden verschiedene Konten zur Verwaltung von Kontakten unterstützt (z. B.: Gmail, Facebook, Last.fm, ...). Die Android-Versionen 2.0 und 2.0.1 sind überholt[2]. Aus diesem Grund ist API Level 7 die frühestmögliche Plattformversion, welche für die Umsetzung der Adressbuchintegration infrage kommt. Eine Unterstützung von Android-Versionen kleiner 2.1 (API Level < 7) ist unnötig, da nur noch etwa 25 %[3] der Anwender diese Versionen einsetzen.

Semantic Web Core Der „Semantic Web Core“ nutzt diese Funktionalität zur Adressbuchintegration nicht. Daher ist er bereits ab Android 1.5[4] (API Level 3) einsetzbar.

[2] Android-Plattform-Versionen: `http://developer.android.com/resources/dashboard/platform-versions.html`, Stand 1. Oktober 2010
[3] Android 1.5: 9,7 %, Android 1.6: 16,4 % und Sonstige: 0,1 %, siehe Fußnote 2
[4] Älteste unterstützte Android-Version, siehe Fußnote 2

5.2 Semantic Web Core

Der Semantic Web Core (Arbeitstitel „Mobile Semantic Web Middleware“) bildet die Grundlage des Social Semantic Web Clients. Er ist für das Laden von Ressourcen aus dem Semantic Web, das Speichern und die Bereitstellung dieser über definierte Schnittstellen zuständig. Die dazugehörigen Objekt-Klassen liegen in dem Java-Paket `org.aksw.msw`.

5.2.1 Klassenübersicht des Semantic Web Cores

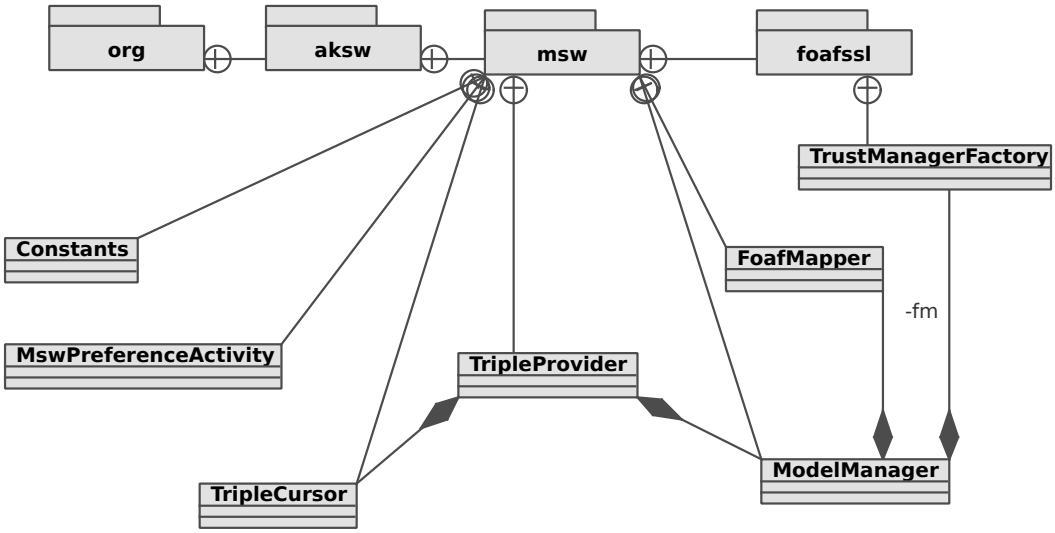

Abbildung 5.2: Klassendiagramm der Mobile Semantic Web Middleware

Abbildung 5.2 zeigt die Klassenübersicht des `org.aksw.msw`-Pakets. Es besteht aus der `Constants`-Klasse, welche wiederkehrend verwendete konstante Werte zusammenfasst. Die Klasse `MswPreferenceActivity` stellt eine Benutzeroberfläche zur Konfiguration des Semantic Web Cores zur Verfügung. Der `TripleProvider` ist ein `ContentProvider`, er stellt über eine Abfrageschnittstelle (Abschnitt 5.2.2) Funktionen bereit, um Ressourcen aus dem Semantic Web abzufragen. Diese Ressourcen werden durch den `ModelManager` verwaltet und gespeichert und in einem `TripleCursor`-Objekt zurückgegeben. Die `FoafMapper`-Klasse bildet das Regelsystem (siehe Abschnitt 5.2.5) und ist damit für die Abbildung des FOAF-Vokabulars (siehe Abschnitt 2.2) auf das Adressbuchschema zuständig. Das Paket `org.aksw.msw.foafssl` beinhaltet die `TrustManagerFactory`, welche für den Aufbau einer sicheren Verbindung über SSL und eine Authentifizierung mittels FOAF+SSL (siehe Abschnitt 2.2 und 6.2.1) nötig ist.

5.2.2 Abfrageschnittstellen der Klasse „TripleProvider“

Für den Semantic Web Core wird eine Klasse `TripleProvider` implementiert, welche die `ContentProvider`-Klasse (Siehe Abschnitt 2.3) erweitert. Der `TripleProvider` ist mit der Authority `org.aksw.msw.tripleprovider` erreichbar und kann über die folgenden Pfade angefragt werden.

Ressource abfragen Die vier Pfade `resource/`, `resource/tmp/*`, `resource/save/*` und `resource/offline/*` sind für die Abfrage einzelner Ressourcen mit deren Eigenschaften vorgesehen. Sie können nur über die `query()`-Methode angefragt werden, andernfalls ist der Rückgabewert `null` beziehungsweise `0`. Die Anfrage wird mit einem `TripleCursor` (siehe Abschnitt 5.2.3) beantwortet, welcher alle Tripel mit der Ressource als Subjekt enthält. Eine Anfrage der hier genannten Pfade wird durch die auf dem mobilen Gerät persistent gespeicherten Ressourcen beantwortet. Ist die angefragte Ressource nicht offline vorhanden, wird sie, je nach Pfad unterschiedlich, bezogen. Der Pfad `resource/tmp/*` lädt eine Ressource temporär nach den Linked Data-Regeln. Es besteht anschließend keine Garantie, dass diese Ressource permanent auf dem mobilen Gerät gespeichert ist. Eine Anfrage des Pfades `resource/save/*` hingegen speichert die Ressource anschließend permanent. Durch Anfrage des Pfades `resource/offline/*` werden nur offline verfügbare Ressourcen zurückgegeben, ist die Ressource nicht vorhanden ergibt dies einen Rückgabewert `null`. Der Standardpfad `resource/` verhält sich wie `resource/save/*`. Wird der Methode `query()` im zweiten Argument (`projection`) ein `String`-Array mit URIs übergeben, werden nur die darin enthaltenen Eigenschaften der angefragten Ressource im Ergebnis ausgegeben. Wenn das dritte Argument auf einen Wert ungleich `null` gesetzt ist, werden alle Eigenschaften der Ressource, außer den im zweiten Argument angegebenen, ausgegeben. Dies gilt auch für das Abfragen von Blank Nodes.

Blank Node abfragen Der Pfad `bnode/*/*` ist eine Übergangslösung. Er kann nur über die `query()`-Methode angefragt werden, andernfalls ist der Rückgabewert `null` beziehungsweise `0`. Es wurde nötig Blank Nodes anhand einer ID zu beziehen, da auf die Ressource-Anfragen (siehe Abschnitt 5.2.2) noch nicht mit der „Concise Bounded Description“ (CBD, Stickler [2005]) der Ressource geantwortet wird. Den ersten Stern (`*`) muss man durch den URI der Ressource ersetzen, an welchem der Blank Node hängt. Der zweite Stern (`*`) muss durch die sich ändernde Jena-AnonId ersetzt werden, welche man durch den Aufruf von `Resource.getId()` auf einer anonymen Ressource erhält. Dieser Pfad wird in Zukunft obsolet sein, sobald eine CBD für die Ressource-Anfrage implementiert ist.

Ressource bearbeiten Die zwei Pfade `resource/addData/*` und `resource/addTriple/*` sind zum Hinzufügen neuer Eigenschaften zu einer Ressource gedacht. Der Stern (`*`)

muss durch die Ressource ersetzt werden, zu welcher die Daten hinzugefügt werden sollen. Die beiden Pfade können über die `insert()` und `update()`-Methode verwendet werden. Der zweite Parameter der Methoden darf nicht `null` sein und muss ein `ContentValues`-Objekt enthalten. Für den Pfad `resource/addTriple/*` muss dieses Objekt Werte für die Schlüssel `subject`, `predicate` und `object` enthalten. Bei der Verwendung des Pfads `resource/addData/*` werden Schlüssel der Form `data<n>`[5] und `mimetype` erwartet, wobei `<n>` durch eine Zahl von 1 bist 15 ersetzt werden muss. Ist der Schlüssel `mimetype` angegeben wird ein gültiger Android-MIME-Type für Adresseinträge erwartet. Siehe dazu die Felder `CONTENT_ITEM_TYPE` der Objekte in der Android-Dokumentation zu `CommonDataKinds`[6].

Aktualisierung von Ressourcen Die beiden Pfade `update/*` und `update/` (der Unterschied liegt in dem Stern, `*`) stoßen eine Aktualisierung (siehe Abschnitt 5.2.4) einer beziehungsweise mehrerer Ressourcen an. Sie können nur über die `update()`-Methode verwendet werden. Der zweite Parameter der Methode darf nicht `null` sein (`new ContentValues()` ist möglich), andernfalls produziert das Android-System einen Ausnahmezustand („Exception“). Wird der Stern durch eine Ressource ersetzt, wird nur diese Ressource aktualisiert, wird an dieser Stelle nichts angegeben, werden alle permanent gespeicherten Ressourcen nach den Linked Data-Regeln (siehe Abschnitt 2.1) erneut angefragt und aktualisiert.

SPARQL Anfrage Die Funktionalität zu dem Pfad `sparql/*` ist nicht implementiert worden. Siehe Abschnitt 7.1 „Ausblick“.

5.2.3 Abfrageschnittstellen der Klasse „TripleCursor“

Der `TripleCursor` ist ein erweiterter `AbstractCursor`[7]. Er enthält die folgenden Spalten:

`_id` Fortlaufende Nummer, eindeutig in jedem Cursor-Objekt

`subject` URI der Subjekt Ressource

`predicate` Die Eigenschaft der Ressource

[5] Android Contacts-Contract Datenspalten: `http://developer.android.com/reference/android/provider/ContactsContract.DataColumns.html`

[6] Android Contacts-Contract Datentypen: `http://developer.android.com/reference/android/provider/ContactsContract.CommonDataKinds.html`

[7] Android-Klasse „AnstractCursor“: `http://developer.android.com/reference/android/database/AbstractCursor.html`

`object` Der entsprechende Wert der Eigenschaft

`predicateReadable` Eine für Menschen lesbare Darstellung der Eigenschaft, in der Regel der letzte Teil des URI

`objectReadable` Eine für Menschen lesbare Darstellung des Werts, siehe auch Name Helper in Abschnitt 5.3.9

`oIsResource` Enthält die Zeichenkette „true“ oder „false“, je nachdem ob das Objekt eine Ressource oder ein Literal ist

`oIsBlankNode` Enthält die Zeichenkette „true“ oder „false“, je nachdem ob das Objekt ein Blank Node ist oder nicht

Die drei Spalten `subject`, `predicate` und `object` bilden zusammen ein RDF-Tripel (siehe Abschnitt 2.1). Die Liste dieser Spalten kann über die Methode `getColumnNames()` abgefragt werden. Die Einträge in den Spalten der einzelnen Datensätze werden über die Methode `getString(int column)` abgefragt. Alle übrigen Methoden wurden nicht implementiert, da sie bei meinen Tests nicht die erwarteten Ergebnisse lieferten.

5.2.4 Spezifikation der Modellverwaltung

Ein Modell wird durch den `ModelManager` (siehe Abschnitt 6.2.1) organisiert. Die Speicherung der Wissensbasen findet in dem Anwendungsverzeichnis[8] auf dem externen Speicher (in der Regel eine SD-Karte) statt. Darin liegt ein Ordner `models` welcher die Unterordner `web`, `inf`, `local` und `cache` enthält. Sie unterscheiden sich folgendermaßen: im Ordner `web` werden Kopien der Onlineversion der Ressourcen aufbewahrt; im Ordner `inf` werden zusätzlich die durch Inferenz entstehenden Eigenschaften gespeichert (siehe unten „Regelsystem“); Ressourcen im Ordner `local` besitzen zusätzliche auf der Mobilplattform erstellte Eigenschaften; der Ordner `cache` ist für die Zwischenspeicherung von Ressourcen vorgesehen.

One-model-per-resource In den Modellordnern wird pro Ressource eine Datei bzw. ein Modell angelegt. Dies bietet die Möglichkeit Ressourcen unabhängig voneinander zu verwalten, da Androjena keine *Named Graphs* (Carroll u. a. [2004]) unterstützt. Bei der Aktualisierung einer Ressource wird das Modell der im System vorhandenen Ressource vollständig geleert. Anschließend wird die Ressource nach den Linked Data-Regeln (siehe Abschnitt 2.1) bezogen und in das vorhandene Modell importiert.

[8] `/sdcard/Android/data/org.aksw.msw/files/`

5.2.5 Spezifikation des Regelsystems

Da sich das FOAF-Vokabular nicht eins zu eins auf das Android-Adressbuchschema abbilden lässt und es auch denkbar ist in Zukunft andere Vokabularien zu unterstützen ist ein möglichst flexibles Regelsystem nötig (vergleiche Abschnitt 3.2.7). Zu diesem Zweck wird der in Androjena enthaltene „Jena 2 Inference support“ verwendet. Um das Regelsystem so flexibel wie möglich zu gestalten sind die Regeln in einer Datei (`/sdcard/Android/data/org.aksw.msw/files/rules.n3`) auf dem externen Speicher abgelegt und können dort nach der „RULEsyntax“[9] bearbeitet werden. Die einzelnen Regeln sollten Abbildungen auf das in Abschnitt 5.3.7 beschriebene Adressbuchvokabular enthalten.

Unabhängig von den beschriebenen Regeln enthalten die ersten beiden Zeilen Informationen zur Aktualisierung der Datei (Listing 5.1). Ist die zweite Zeile auf `yes` gesetzt wir die Datei überschrieben sobald eine neuere Version installiert wurde, als die in der ersten Zeile anstelle des `x` Angegebene. Ein Beispiel einer Regel wird in Abschnitt 6.2.2 vorgestellt.

```
1 # version=x
2 # update=yes
```

Listing 5.1: Versionierung der Regeldatei `rules.n3` (Auszug)

5.3 FOAF/WebID Provider und Browser

Der FOAF/WebID Provider und Browser (Arbeitstitel „Mobile Social Semantic Web Middleware“) vereinigt die sozialen Komponenten, welche auf dem Semantic Web Core aufsetzen. Die dazugehörigen Objektklassen sind in dem Java-Paket `org.aksw.mssw` zusammengefasst.

5.3.1 Klassenübersicht des FOAF/WebID Providers und Browsers

Abbildung 5.3 zeigt die Klassenübersicht des `org.aksw.mssw`-Pakets. Die Klasse `FirstRun` stellt einen Konfigurationsassistenten zur Verfügung, welcher beim ersten Start des Browsers oder dem erstmaligen Erstellen eines Adressbuchaccounts aufgerufen wird (siehe Abschnitt 5.3.8). Die Klasse `MsswPreferenceActivity` stellt eine Benutzeroberfläche zur Konfiguration des FOAF/WebID Providers und Browsers zur Verfügung und ruft die Konfiguration des Semantic Web Cores auf. Die beiden Klassen `Constants`

[9] Jena Regelsystem: `http://jena.sourceforge.net/inference/#RULEsyntax`

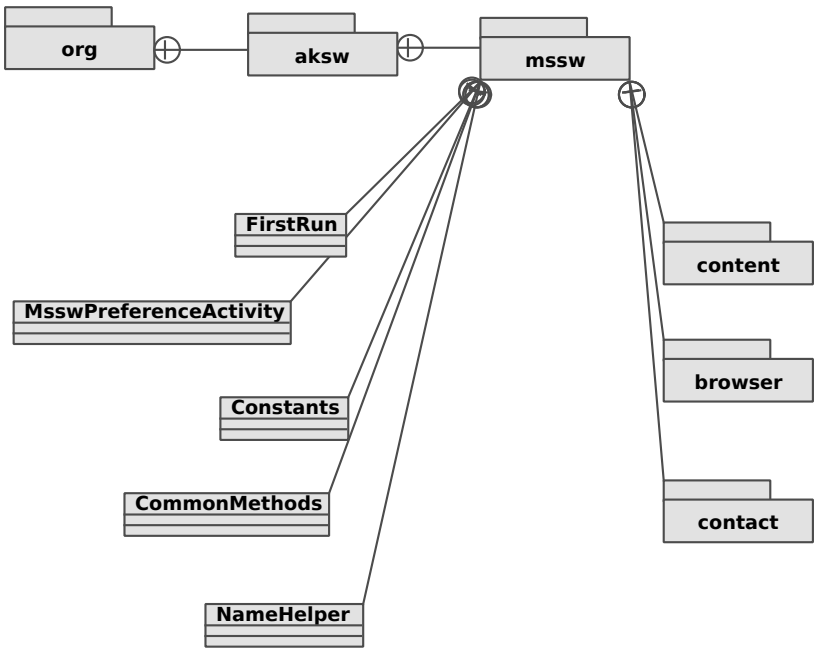

Abbildung 5.3: Klassendiagramm des Mobile Social Semantic Web Middleware

und `CommonMethods` fassen wiederkehrend verwendete konstante Werte bzw. Methoden zusammen. Der `NameHelper` liefert für Menschen lesbare Namen zu angegebenen Ressourcen (siehe Abschnitt 5.3.9). Außerdem sind drei Unterpakete enthalten, eine Abfrageschnittstelle für soziale Semantic Web Funktionen in dem Paket `org.aksw.mssw.content` (siehe Abschnitt 5.3.2), die Adressbuchintegration in dem Paket `org.aksw.mssw.contact` (siehe Abschnitt 5.3.6) und eine beispielhafte Benutzerschnittstelle in dem Paket `org.aksw.mssw.browser` (siehe Abschnitt 5.3.5).

5.3.2 Klassenübersicht des FOAF-Providers

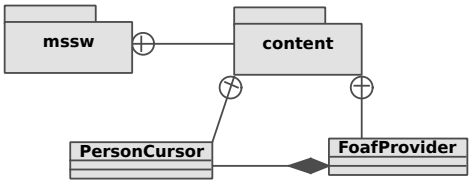

Abbildung 5.4: Klassendiagramm FoafProvider

Der FOAF-Provider (`FoafProvider`) ist in dem Java-Paket `org.aksw.mssw.content` zusammengefasst. Abbildung 5.4 zeigt die Klassenübersicht des `org.aksw.mssw.content`-Pakets. Um im Social Semantic Web zu navigieren wird eine Klasse `FoafProvider` im-

plementiert. Bei Anfrage des `FoafProviders` liefert er Ergebnisse in `PersonCursor` Objekten oder reicht die Antworten des `TripleProviders` (`TripleCursor`, siehe Abschnitt 5.2.3) durch.

5.3.3 Abfrageschnittstellen des FOAF-Providers

Der `FoafProvider` ist ebenso, wie der `TripleProvier` als Schnittstelle für andere Anwendungen vorgesehen. Im Gegensatz zum `TripleProvier` bietet er spezielle Funktionen eines sozialen Netzwerkes an. Er erweitert die `ContentProvider`-Klasse (Siehe Abschnitt 2.3), ist mit der Authority `org.aksw.mssw.foafprovider` erreichbar und kann über die folgenden Pfade angefragt werden. Bei allen Anfragen über die `query()`-Methode wird das zweite Argument an den `TripleProvider` weitergegeben.

Alle Eigenschaften Alle Eigenschaften einer Person können mittels der `query()`-Methode und dem Pfad `person/*` abgefragt werden. Es wird die Antwort des `TripleProviders` weitergereicht, die Anfrage entspricht damit einer direkten Anfrage an den `TripleProvider` mit dem Pfad `resource/*`. Der Pfad `me` ist eine Kurzform des Pfads `person/*`, wobei der Stern (`*`) durch die konfigurierte Web ID des Benutzers ersetzt wird.

Profilseite Mit dem Pfad `person/mecard/*` können die Eigenschaften einer Person über die `query()`-Methode abgefragt werden. Es werden alle Eigenschaften der Person, außer der `foaf:know` Eigenschaft und den Eigenschaften des Relations-Vokabulars (Davis u. Jr [2010]), ausgegeben Es wird die Antwort des `TripleProviders` weitergereicht. Der Pfad `me/mecard` ist wieder eine Kurzform für `person/mecard/*` (siehe „Alle Eigenschaften“).

Kontaktliste Mit dem Pfad `person/friends/*` kann die Liste der Kontakte einer Person über die `query()`-Methode abgefragt werden. Die Antwort besteht aus einem `PersonCursor` (siehe unten). Der Pfad `me/friends` ist wieder eine Kurzform für `person/friends/*` (siehe „Alle Eigenschaften“).

Foto Der Pfad `person/picture/*` kann über die `query()`-Methode abgefragt werden, um damit eine bildhafte Darstellung einer Person zu erhalten. Es wird die Antwort des `TripleProviders` weitergereicht, die Anfrage entspricht damit einer direkten Anfrage an den `TripleProvider` mit dem Pfad `resource/*` und der Projektion `http://xmlns.com/foaf/0.1/depiction`.

Profil finden Der Pfad `search/*` soll ebenfalls über die `query()`-Methode abgefragt werden, um eine oder mehrere Profile zu erhalten, die auf eine Sucheingabe passen. Die Sucheingabe wird URL-encodiert anstelle des Sterns (`*`) an den Pfad angehängt. Momentan wird nur die direkte Eingabe eines URI unterstützt (siehe Abschnitt 7.1), das Ergebnis enthält genau dieses Profil oder ist leer in dem Fall, dass der URI nicht in ein RDF-Dokument aufgelöst werden kann. Die Antwort ist in einem `PersonCursor` (siehe unten) zusammengefasst.

Freund hinzufügen Neue Freundschaften werden zu dem Profil des Benutzers mit der `update()`- oder `insert()`-Methode und dem Pfad `me/friend/add` hinzugefügt. Die als neuer Freund hinzuzufügende Ressource wird über ein `ContentValues`-Objekt mit dem Schlüssel „webid“ an zweiter Stelle der Methode übergeben. Es wird die `foaf:knows` Relation verwendet. Die neuen Freundschaftsrelationen werden in einem lokalen Modell in der Modellverwaltung (siehe Abschnitt 5.2.4) angelegt, allerdings nicht in das entsprechende Modell auf dem Webserver zurückgeschrieben. Siehe Abschnitt 7.1 „Ausblick“.

5.3.4 Abfrageschnittstellen der Klasse „PersonCursor“

Der `PersonCursor` fasst mehrere Web IDs von Kontakten oder anderen Personen zusammen. Er ist ein erweiterter `AbstractCursor`[10] mit den folgenden Spalten:

`_id` Fortlaufende Nummer, eindeutig in jedem Cursor-Objekt

`webid` Web ID der Person

`relation` Die Eigenschaft, durch welche der Benutzer mit dieser Person verbunden ist

`name` Name der Person

`relationReadable` Eine für Menschen lesbare Darstellung der Eigenschaft

`picture` Der URI einer bildhaften Darstellung der Person (Platzhalter, wird noch nicht verwendet)

Zusammen mit dem URI des Benutzers (Subjekt) bilden die Spalten `relation` (Prädikat) und `webid` (Objekt) ein Tripel.

[10] Android-Klasse „AnstractCursor“: `http://developer.android.com/reference/android/database/AbstractCursor.html`

5.3.5 Klassenübersicht des FOAF-Browsers

Der FOAF Browser ist die beispielhaft implementierte Benutzeroberfläche zur Bedienung der sozialen Funktionalität des Semantic Web Clients. Er bietet die Möglichkeit Profile und Kontaktlisten anzuzeigen, neue Beziehungen zur eigenen Web ID hinzuzufügen und beliebige Web IDs anzuzeigen. Abbildung 5.5 zeigt die Klassenübersicht des `org.aksw.mssw.browser`-Pakets. Es besteht aus einer Klasse `Browser`, welche wahlweise eine der Klassen `BrowserMeCard`, `BrowserContacts` oder `BrowserBrowse` in einem Tab anzeigt. Diese drei Klassen rufen für die Menüverwaltung die Klasse `MenuManager` auf.

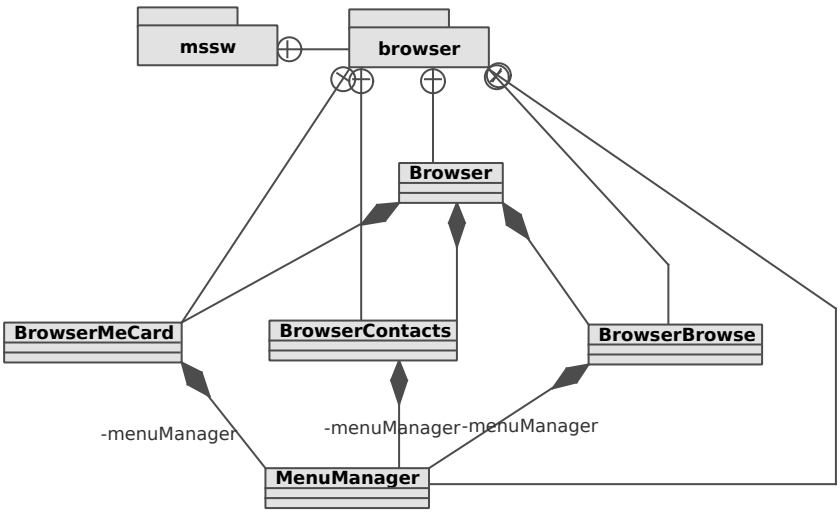

Abbildung 5.5: Klassendiagramm Browser

Intents Der FOAF-Browser nutzt `Intents`, um Aktionen durchzuführen (vergleiche Abschnitt 2.3). Die Definierten `Intents` sind: `org.aksw.mssw.VIEW_WEBID`, um eine Web ID anzuzeigen; `org.aksw.mssw.ADD_WEBID`, um einen Kontakt zur Web ID des Benutzers hinzuzufügen und `org.aksw.mssw.ERROR` für das Übermitteln von Fehlern. Die ersten Beiden erwarten einen `android.net.Uri` mit einer Web ID als „Intent data URI“. Der Intent `org.aksw.mssw.ERROR` erwartet `Strings` als „extra data“ mit den Schlüsseln `error_titel` und `error_message`.

5.3.6 Klassenübersicht der Adressbuchintegration

Die Adressbuchintegration fügt die Kontakte des Benutzers in das Systemadressbuch ein, um das Social Semantic Web besser in die bestehende mobile Plattform zu integrieren. Abbildung 5.6 zeigt die Klassenübersicht des `org.aksw.mssw.contact`-Pakets. Es besteht aus einem `AccountAuthenticator`, welcher mit Hilfe des `AccountAuthenticator-`

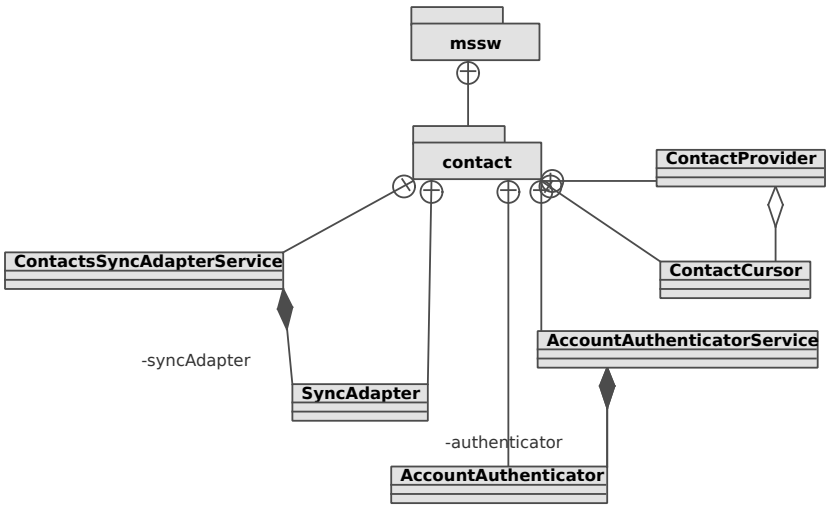

Abbildung 5.6: Klassendiagramm Adressbuchintegration

`Service`s einen Android-Web ID-Kontotyp zur Verfügung stellt. Wird in den Android-Einstellungen ein Konto des Web ID-Kontotyps angelegt und synchronisiert, wird der `SyncAdapter` aufgerufen, welcher mit Hilfe des `ContactsSyncAdapterService`s die Web-IDs in das Adressbuch schreibt. Dazu fragt der `ContactsSyncAdapterService` den `ContactProvider` an, welcher mit einem `ContactCursor` antwortet.

Abfrageschnittstellen Der `ContactProvider` besitzt nur einen Pfad `data/*`, welcher über die `query()`-Methode angefragt wird. Der Stern (`*`) wird durch die Web ID eines Kontakts ersetzt, um dessen Adressbuchdaten zu erhalten. Die Anfrage wird mit einem `ContactCursor`-Objekt beantwortet.

Das „ContactCursor“-Objekt Der `ContactCursor` ist ein erweiterter `AbstractCursor`[11] und dem `TripleCursor` sehr ähnlich (vergleiche Abschnitt 5.2.3). Er besitzt folgende Spalten:

`_id` Fortlaufende Nummer, eindeutig in jedem Cursor-Objekt

`subject` URI der Subjekt-Ressource

`predicate` Die Eigenschaft der Ressource

`object` Der entsprechende Wert der Eigenschaft

[11] Android-Klasse „AnstractCursor“: `http://developer.android.com/reference/android/database/AbstractCursor.html`

`oIsResource` Enthält die Zeichenkette „true“ oder „false“, je nachdem ob das Objekt eine Ressource oder ein Literal ist

`oIsBlankNode` Enthält die Zeichenkette „true“ oder „false“, je nachdem ob das Objekt ein Blank Node ist oder nicht

Er enthält nur Tripel entsprechend dem *Adressbuchschema* (siehe unten) und ist für den internen Gebrauch zur Adressbuchintegration vorgesehen.

5.3.7 Beschreibung des Adressbuchvokabulars

Die Adressbuchdaten werden im Android-System in einer Relationalen Datenstruktur abgelegt. Sie besteht aus den Tabellen `Data`, `RawContacts` und `Contacts` [Meier, 2010, 240 ff]. Hier ist nur die Tabelle `Data`[12] relevant. Jede Zeile in dieser Tabelle besteht unter anderem aus einem Feld `_ID`, zur Identifikation der Zeile; einem Feld `MIMETYPE`, welches den Typ der Zeile angibt; einem Feld `RAW_CONTACT_ID` zur Identifikation eines Kontakts; 15 Datenfeldern (`DATA1` bis `DATA15`) und vier frei belegbaren Zusatzfeldern für den Synchronisationsstatus (`SYNC1` bis `SYNC4`). Die Datenfelder werden je nach Typ der Zeile unterschiedlich belegt. Diese Belegung wird in einer Klassenstruktur abgebildet. In der Klasse `ContactsContract.CommonDataKinds` des Pakets `android.provider` ist für jeden Datentyp (z. B. `StructuredName`, `Email` und `Phone`) eine Unterklasse vorhanden. Jede dieser Klassen besitzt ein Feld `CONTENT_ITEM_TYPE`, welches den MIME-Type (z. B. `vnd.android.cursor.item/name`, `vnd.android.cursor.item/email_v2` und `vnd.android.cursor.item/phone_v2`) enthält. Außerdem sind weitere Felder mit sprechenden Namen für die Datenfelder vorhanden. So entsprechen `StructuredName.DISPLAY_NAME`, `Email.DATA`, `Phone.NUMBER` und `Data.DATA1` dem ersten Datenfeld `"data1"`.

Für die einfache Abbildung von RDF-Eigenschaften auf die System spezifische Datenbankstruktur des Adressbuchs wurde ein RDF-Vokabular entworfen. Die Struktur aus Klassen wird nahezu eins zu eins in dem Adressbuchvokabular repräsentiert. Jeder URI des Adressbuchvokabulars beginnt mit `http://ns.aksw.org/Android/` (wird mit `android:` abgekürzt). An diesen URI wird der Klassenname angehängt (z. B. `android:ContactsContract.CommonDataKinds.StructuredName`), um eine Klasse zu beschreiben. Die Klassennamen können ebenfalls abgekürzt werden (z. B. `name:`), um die Felder leichter zu beschreiben. Bei der Definition der Abkürzung wird am Ende des Klassennamens ein Punkt (`.`) ergänzt, um das Feld abzutrennen. Ein Feld (z. B. `name:DISPLAY_NAME`) dient als Eigenschaft. Der Wert der Eigenschaft entspricht dem Wert des Feldes.

[12] Android-Contact-Contract Datentabelle: `http://developer.android.com/reference/android/provider/ContactsContract.Data.html`

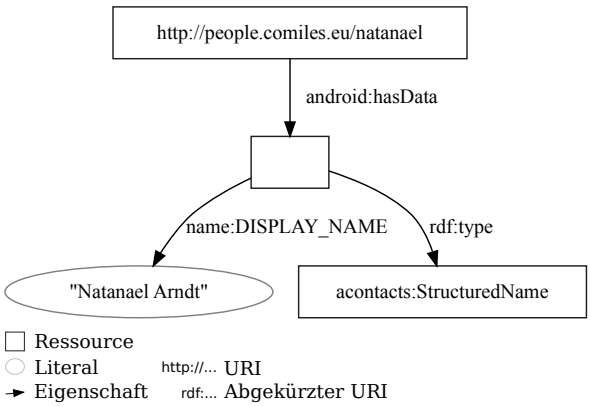

Abbildung 5.7: Nutzung des Adressbuchvokabular in einem Beispiel-RDF-Graphen

Alle Felder einer einzelnen Spalte werden zusammen an eine Ressource gehängt, diese ist in der Regel ein Blank Node. Alle Blank Nodes werden mit der Eigenschaft `android:hasData` an die entsprechende Web ID gehängt. Dies kann so aussehen wie in Abbildung 5.7 dargestellt.

5.3.8 Konfigurationsassistent

Der Konfigurationsassistent ist eine Schnittstelle, welche den Benutzer durch die Konfiguration der nötigen Parameter für den Betrieb des Social Semantic Web Clients führt. Überprüfung ob der Semantic Web Core installiert ist, sonst wird eine Warnung angezeigt. Abfrage der Web ID des Benutzers. Wenn ein Clientzertifikat für FOAF+SSL vorhanden ist wird das Passwort der Zertifikats abgefragt.

5.3.9 Auflösung von URIs durch die Klasse „NameHelper“

Die Klasse `NameHelper` liefert verständliche Namen für eine Ressource. In diesem Fall für Web IDs. Er nimmt einzelne oder eine Liste von Ressourcen in Form von URIs entgegen und beantwortet diese mit dem Wert einer der folgenden Eigenschaften: `foaf:name`, `sioc:name`, `rdfs:label`, `foaf:nick` oder `foaf:surname`. Die jeweils vorderen Eigenschaften werden gegenüber den hinteren bevorzugt. Ist keine dieser Eigenschaften vorhanden wird der vollständige URI zurückgegeben. Eine Beantwortung mit dem letzten Teil des URI ist in diesem Fall nicht brauchbar, da sehr viele Web IDs auf `me` enden und somit keine Unterscheidung möglich ist.

6 Implementierung

Der Social Semantic Web Client besteht aus zwei Android-Java-Projekten, „Mobile Semantic Web“ (`org.aksw.msw`, Abschnitt 6.2, Spezifikation und Aufbau in Abschnitt 5.2) und „Mobile Social Semantic Web“ (`org.aksw.mssw`, Abschnitt 6.3, Spezifikation und Aufbau in Abschnitt 5.3).

6.1 Konstante Werte

Beide Projekte besitzen eine Klasse `Constants`. In dieser Klasse werden häufig verwendete und von der Implementierung an feststehende Werte gesammelt. Die Sammlung an einem zentralen Ort bietet die Möglichkeit der leichten Änderung der Werte, falls dies in einer späteren Version nötig wird.

6.2 Mobile Semantic Web Middleware

Die Mobile Semantic Web Middleware (oben auch „Semantic Web Core“ genannt) soll Daten aus dem Semantic Web für andere Anwendungen zur Verfügung stellen. Daher bildet ein Content-Provider (vergleiche Abschnitt 2.3) das Kernstück der Middleware, der `TripleProvider`. Er ist die einzige für andere Anwendungen ansprechbare Schnittstelle der Mobile Semantic Web Middleware. Der `TripleProvider` nimmt die Anfragen entgegen und ruft die entsprechenden Methoden der Modellverwaltung auf. Die daraus resultierenden Ergebnisse der Modellverwaltung verpackt er in `Cursor`-Objekte. Die Abtrennung der Anfrageschnittstelle von der Modellverwaltung bietet die Möglichkeit in Zukunft andere Anfrageschnittstellen zu implementieren oder eine bessere Modellverwaltung einzusetzen.

```
1  SSLSocketFactory socketFactory = TrustManagerFactory.getFactory(
     keyFile, password);
2
3  if (socketFactory != null) {
4    HttpsURLConnection.setDefaultSSLSocketFactory(socketFactory);
5
6    URLConnection conn = new URL(url).openConnection();
7    conn.setRequestProperty("accept", Constants.REQUEST_PROPERTY);
8    conn.connect();
9
10   InputStream iStream = conn.getInputStream();
11   read(url, model, iStream);
12   iStream.close();
13 }
```

Listing 6.1: Aufbau einer FOAF+SSL-Verbindung in Java

6.2.1 Implementierung der Modellverwaltung

Die Verwaltung der RDF-Wissensbasen ist in der Klasse `ModelManager` implementiert. Die Ressourcen bzw. Modelle werden mit der Methode `Model.read()` des Jena-Frameworks gemäß den Linked Data-Regeln über HTTP geladen. Wenn ein Clientzertifikat installiert ist werden Ressourcen mit FOAF+SSL-Authentifizierung (siehe unten „FOAF+SSL“) geladen. Es wurden keine Geschwindigkeitstests der Modellverwaltung vorgenommen, da die Reaktionszeiten zu sehr von der Internetverbindung abhängen. Die Funktionalität der Modellverwaltung wurde durch den FOAF-Browser und die Adressbuchintegration überprüft.

FOAF+SSL Der Import der Modelle über eine FOAF+SSL-Verbindung wird realisiert, indem der `Model.read()`-Methode zusätzlich zu dem URI ein `InputStream` übergeben wird. Der `InputStream` entsteht aus einer HTTPS-Verbindung, welche mit dem FOAF+SSL-Clientzertifikat aufgebaut wurde. Listing 6.1 zeigt den Aufbau einer FOAF+SSL-Verbindung, es ist ein gekürzter Ausschnitt aus der Klasse `org.aksw.msw.ModelManager`. Die `TrustManagerFactory` (Zeile 1 und 4) erstellt `SSLSocketFactory`s für `TLS`-Verbindungen[1] mit einem `X509`-Clientzertifikat, wie es für eine FOAF+SSL-Verbindung nötig ist (Story u. a. [2009]). Die Klassen `HttpsURLConnection` und `HttpURLConnection` erweitern beide die Klasse `URLConnection`. Wird eine URL-Verbindung mit dem `https:`-Schema aufgebaut wird automatisch ein `HttpsURLConnection`-Objekt erstellt. In Zeile 7 wird der Verbindung der Accept-Header übergeben, welcher den angefragten Server dazu auffordert mit RDF-Daten zu antworten. Die Methode `read(url, model, iStream)` in Zeile 11 liest RDF-Tripel mit dem Subjekt `url` aus dem Datenstrom `iStream` in das

[1] TLS ist eine Weiterentwicklung des SSL, umfasst somit sowohl TLS als auch SSL

angegebene Modell `model`. Ist kein Datenstrom angegeben wird eine unverschlüsselte Verbindung aufgebaut.

6.2.2 Implementierung des Regelsystems

Die Klasse `FoafMapper` stellt das Regelsystem dar. Sie liest aus der Regeldatei[2] eine Liste von Regeln der Klasse `com.hp.hpl.jena.reasoner.rulesys.Rule` ein. Diese Datei wird in einen neuen Reasoner der Klasse `com.hp.hpl.jena.reasoner.Reasoner` geladen. Über die Methode `map()` wird einem `FoafMapper`-Objekt ein Modell übergeben. Die Methode gibt anschließend das Modell, zuzüglich der neu gewonnenen Aussagen, zurück.

Aktualisierung der Regeldatei Um es jedem versierten Nutzer zu ermöglichen eigene Abbildungen von Eigenschaften auf das Adressbuchvokabular vorzunehmen wird die Datei mit den Abbildungsregeln auf den externen Speicher des Android-Systems (in der Regel eine SD-Karte) geschrieben. Diese Datei bleibt allerdings auch über die Aktualisierung der Anwendung hinaus dort bestehen (sieh auch Abschnitt 5.2.5). Das kann in machen Fällen erwünscht sein, es können aber keine neuen Abbildungen der Entwickler hinzukommen. Daher wird beim Start des Regelsystems überprüft, ob die Datei bereits vorhanden ist. Ist die Datei nicht vorhanden, wird die im Anwendungspaket enthaltene Version an die vorgesehene Stelle geschrieben. Ist die Datei vorhanden werden die ersten beiden Zeilen der Datei gelesen (siehe Listing 5.1). Steht in der ersten Zeile eine kleinere Versionsnummer, als die des installierten Anwendungspakets, und gleicht die zweite Zeile der in Listing 5.1, wird die Datei mit der im Anwendungspaket enthaltenen Version überschrieben. Andernfalls bleibt sie unverändert.

Beispiel einer Regel Listing 6.2 zeigt eine einfache Regel[3], mit einem Namen (Zeile 1), einer Prämisse (Zeile 2) und einer Schlussfolgerung (Zeilen 4-6). Existiert eine Aussage (Tripel) mit beliebigem Subjekt und Objekt und der Eigenschaft `foaf:name`, wird das Subjekt durch die Variable `?s`, das Objekt durch die Variable `?o` und zusätzlich ein neuer Blank Node durch die Variable `?d` gebunden. Dann folgt aus dieser Aussage, dass das Subjekt (`?s`) einen Datensatz (`?d`) vom Typ (`rdf:type`) `acontacts:StructuredName` und mit der Eigenschaft `name:DISPLAY_NAME` mit dem Objekt (`?o`) als Wert besitzt (`android:hasData`).

[2] `/sdcard/Android/data/org.aksw.msw/files/rules.n3`

[3] Alle jeweils aktuell vordefinierten Regeln können durch Abfrage des URI `http://code.google.com/p/mssw/source/browse/msw/res/raw/defaultmapping` eingesehen werden.

```
1 [name:
2   (?s foaf:name ?o), makeTemp(?d)
3   ->
4   (?s android:hasData ?d),
5   (?d rdf:type acontacts:StructuredName),
6   (?d name:DISPLAY_NAME ?o)
7 ]
```

Listing 6.2: Beispiel einer Jena-Regel, Abbildung der Eigenschaft `foaf:name` auf das Adressbuchvokabular

6.3 Mobile Social Semantic Web Middleware

6.3.1 Implementierung des FOAF-Providers

Der `FoafProvider` ist ebenso wie der `TripleProvider` ein `ContentProvider` (siehe Abschnitt 2.3), er ist ebenso mit wenig eigener Funktionalität ausgestattet. Er nimmt lediglich die Anfragen entgegen und bezieht die nötigen Daten mit geänderten Anfragen aus dem `TripleProvider`. Dabei reicht er die erhaltenen `Cursor`-Objekte weiter. Lediglich Kontaktlisten und Suchergebnisse „packt er um“ in `PersonCursor`-Objekte.

6.3.2 Implementierung des FOAF-Browsers

Der FOAF-Browser ist die Benutzerschnittstelle des Social Semantic Web Clients. Er ist durch eine Sammlung von `Activity`-Objekten gebaut, welche als einzelne Bildschirmseiten dem Benutzer dargestellt werden. Die Hauptansicht ist die Klasse `Browser`, eine `TabActivity`. Der `Browser` nimmt als zentrale Stelle mit der Methode `onNewIntent` alle Intents (siehe Abschnitt 5.3.5) entgegen. Er bearbeitet sie in der Methode `handleIntent` indem er den entsprechend neuen Status in ein von allen Ansichten gemeinsam genutztes Objekt `SharedPreferences` schreibt. Anschließend bringt er die zugehörige Ansicht in den Vordergrund. Die Ansichten `BrowserMeCard`, `BrowserContacts` und `BrowserBrowse` implementieren das Interface `OnSharedPreferenceChangeListener` wodurch sie über Änderungen des `SharedPreferences`-Objekts informiert werden und ihr Erscheinen anpassen. Sie fragen dazu den `FoafProvider` entsprechend an und übergeben den resultierenden `Cursor` in einem `SimpleCursorAdapter` an einen `ListView`, welcher den Inhalt zur Anzeige bringt.

Abbildung 6.1 zeigt den FOAF-Browser mit der Profilansicht (`BrowserMeCard`). Im oberen Bereich wird der Name der Person angezeigt, welcher durch den `NameHelper` bezogen wird. Darunter sind alle Eigenschaften der Web ID angezeigt, abzüglich der `android:hasData` Eigenschaft, der `foaf:knows` Eigenschaft und der Eigenschaften des

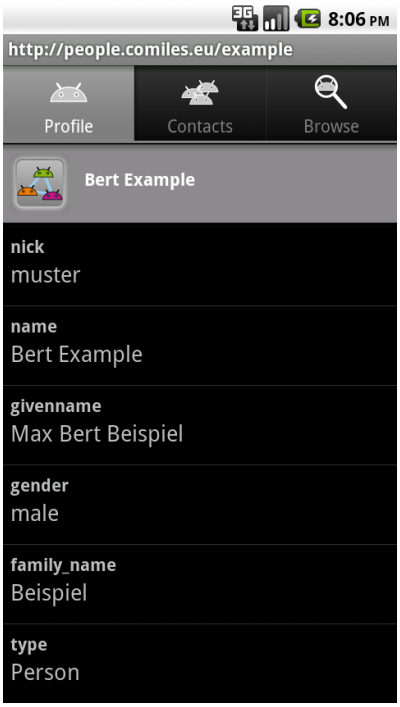

Abbildung 6.1: Bildschirmfoto der Profilansicht

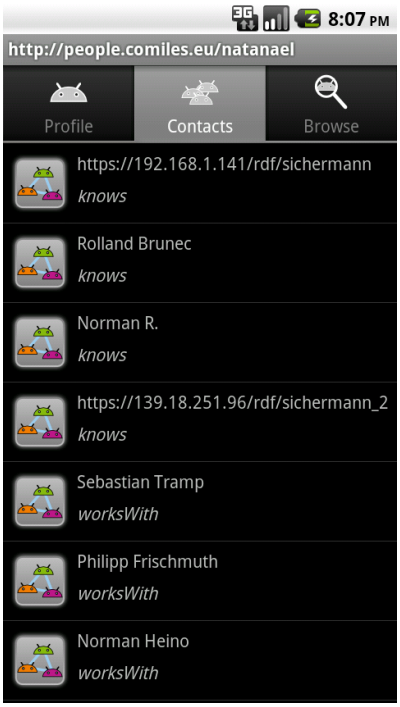

Abbildung 6.2: Bildschirmfoto der Kontaktliste

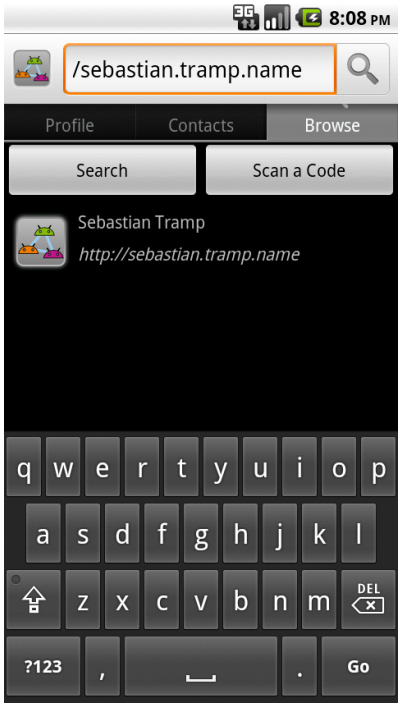

Abbildung 6.3: Bildschirmfoto der Suchfunktion

Relations-Vokabulars. Abbildung 6.2 zeigt den FOAF-Browser mit der Kontaktliste (`BrowserContacts`). Sie listet die Kontakte der im Profil angezeigten Person auf. Jeder Eintrag der Liste besteht aus dem Namen, sofern er durch den `NameHelper` bezogen werden konnte, und der abgekürzten Relation. Abbildung 6.3 zeigt den FOAF-Browser mit der Suchansicht (`BrowserBrowse`). Durch klicken des Knopfes „Search“ oder der Suchtaste des Telefons wird die Android-Suchleiste angezeigt. Sie akzeptiert momentan lediglich die Eingabe einer Web ID (siehe Abschnitt 7.1). Alternativ kann auch eine Web ID durch einen QR-Code eingelesen werden. Das Ergebnis wird in der darunterliegenden Liste angezeigt. Um die `Activity` auf der Suchtaste zu registrieren müssen in der Datei `AndroidManifest.xml` die Eintragungen aus Listing 6.3 getätigt werden.

```
1  <manifest ... >
2    <application ... >
3      <meta-data android:name="android.app.default_searchable"
         android:value=".browser.Browser" />
4      <activity android:name=".browser.Browser" ... >
5        <intent-filter>
6          <action android:name="android.intent.action.SEARCH" />
7        </intent-filter>
8        <meta-data android:name="android.app.searchable" android:
           resource="@xml/searchable" />
9      </activity>
10   </application>
11 </manifest>
```

Listing 6.3: Einbinden der Android-Suchfunktion in der Datei `AndroidManifest.xml`

6.3.3 Implementierung der Adressbuchintegration

Für die Integration der Web ID in das Android-Systemadressbuch müssen zwei Services in der Datei `AndroidManifest.xml` registriert werden (Listing 6.4). Der erste Service (Zeile 3-8) erstellt einen neuen Accounttyp (Abbildung 6.4 und 6.5).

```
1  <manifest ... >
2    <application ... >
3      <service android:name=".contact.AccountAuthenticatorService"
         android:exported="true" android:process=":auth">
4        <intent-filter>
5          <action android:name="android.accounts.
             AccountAuthenticator" />
6        </intent-filter>
7        <meta-data android:name="android.accounts.
           AccountAuthenticator" android:resource="@xml/authenticator
           " />
8      </service>
9      <service android:name=".contact.ContactsSyncAdapterService"
         android:exported="true">
10       <intent-filter>
```

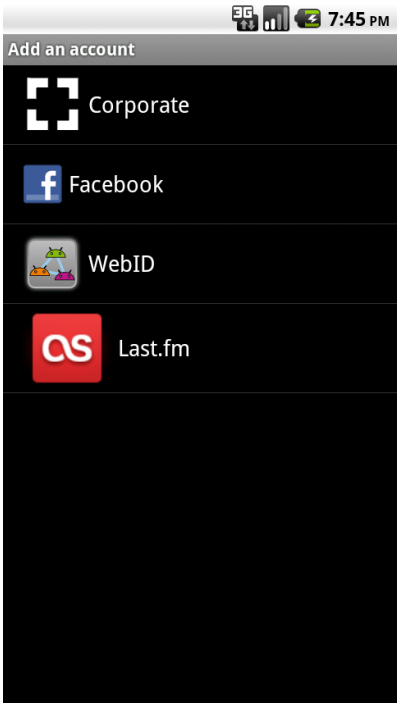

Abbildung 6.4: Bildschirmfoto der Accounttypenauswahl

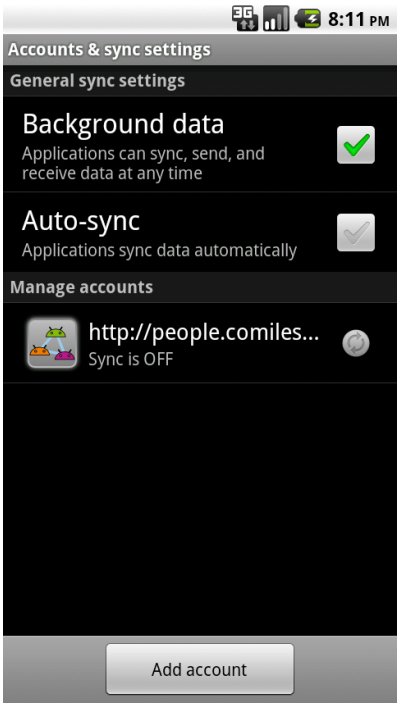

Abbildung 6.5: Bildschirmfoto der Accountverwaltunge

```
11          <action android:name="android.content.SyncAdapter" />
12        </intent-filter>
13        <meta-data android:name="android.content.SyncAdapter"
            android:resource="@xml/sync_contacts" />
14      </service>
15    </application>
16  </manifest>
```

Listing 6.4: Definition des Android-Kontotyps und der Synchronisationsschnittstelle in der Datei `AndroidManifest.xml`

Abbildung 6.6 zeigt die Synchronisation von Kontakten in das Adressbuch. Der dafür nötige Synchronisationsadapter wird in Zeile 9-14 registriert. Er fügt die neuen Kontakte (Abbildung 6.7) mit den Eigenschaften der Web ID (Abbildung 6.8 und 6.9) in die Adressbuchtabelle (siehe Abschnitt 5.3.7) ein.

Synchronisationsprozess Wie in Abschnitt 4.2 beschrieben, wäre es denkbar für den Synchronisationsprozess die von Schandl (Schandl [2010]) beschriebene Technik zu verwenden. Zu diesem Zeitpunkt, da mir diese Idee kam, war die Arbeit am Synchronisationsprozess bereits begonnen. Daher fiel die Wahl auf eine einfachere, wenn auch sicher ineffektivere Methode. Mit entsprechendem Aufwand wäre eine Neuimplementierung

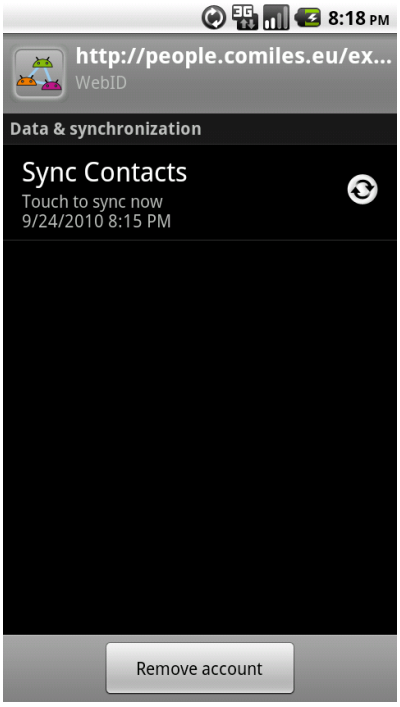

Abbildung 6.6: Bildschirmfoto der Adressbuchsynchronisation

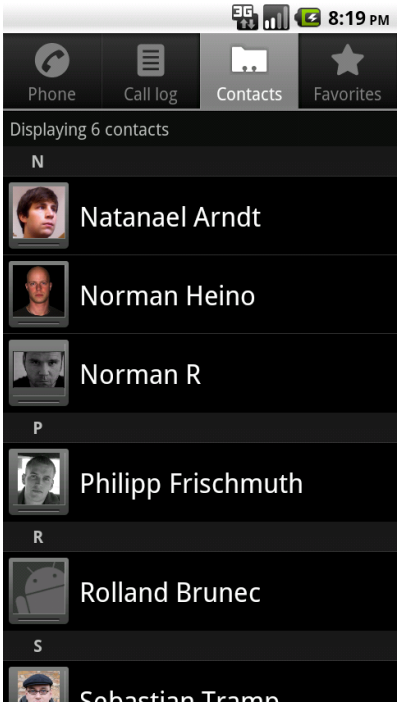

Abbildung 6.7: Bildschirmfoto des Adressbuchs

des Synchronisationsprozesses allerdings denkbar. Der jetzt verwendete Synchronisationsprozess lädt zuerst die Kontaktliste des Benutzers aus dem `FoafProvider`. Das Ergebnis wird mit der Liste der Kontakte im Adressbuch verglichen. Anschließend wird unterschieden, ob bereits ein Eintrag zu der Web ID im Adressbuch vorhanden ist oder nicht. Im ersten Fall wird der bestehende Kontakteintrag aktualisiert, im zweiten Fall ein neuer Kontakteintrag hinzugefügt. In beiden Fällen werden alle durch `android:hasData` mit der Web ID verbundenen Ressourcen mit ihren Eigenschaften vom `ContactProvider` angefordert.

Übersetzen des Adressbuchvokabulars Alle Ressourcen und Eigenschaften des Adressbuchvokabulars (siehe Abschnitt 5.3.7) entsprechen Feldern von Objekten des Android-Frameworks. Diese Objekte werden mit Hilfe der Java-Methode `java.lang.Class.forName()` des Java-Reflection-API in Objekte umgewandelt, um die entsprechenden Felder zu lesen.

Um einen neuen Kontakt anzulegen, wird diese `android:hasData`-Datenstruktur in eine neue „Einfügen“-Operation übertragen und in die Adressbuchtabelle geschrieben. Für einen vorhandenen Kontakt müssen allerdings erst alle bereits enthaltenen Einträge mit

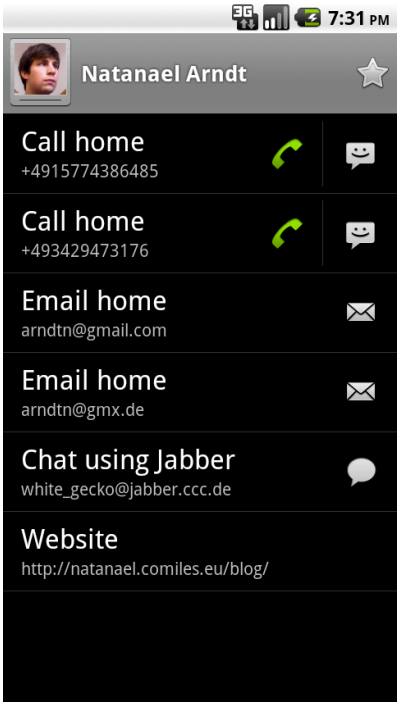

Abbildung 6.8: Bildschirmfoto der Kontaktansicht

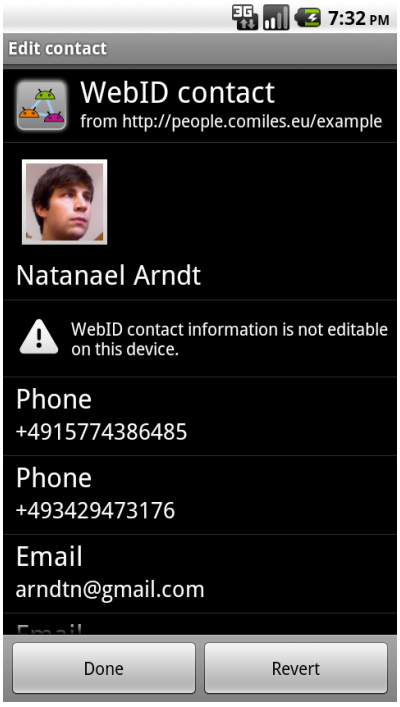

Abbildung 6.9: Bildschirmfoto zeigt Kontakttyp (Web ID)

der `android:hasData`-Datenstruktur verglichen werden und anschließend die fehlenden Eigenschaften ergänzt werden.

6.3.4 Auflösung von URIs durch die Klasse „NameHelper“

Der `NameHelper` besitzt eine `HashMap`, welche `String`s auf `String`s abbildet. Der Schlüssel ist jeweils der URI einer Ressource und der Wert beinhaltet den besten zu findenden Namen. Bei der Anfrage eines Namens für eine Ressource fragt der `NameHelper` als erstes diese Liste ab. Wenn sie die entsprechende Ressource nicht besitzt, startet der `NameHelper` in einem neuen Thread eine Anfrage an den `TripleProvider` nach der Ressource, mit den Namenseigenschaften (siehe Abschnitt 5.3.9) als Projektion. Wenn der Thread nicht innerhalb von 10 Sekunden beendet ist beantwortet der `NameHelper` die Anfrage mit dem URI der Ressource. Bei einer späteren Anfrage ist es sehr wahrscheinlich, dass der Thread beendet wurde und ein Ergebnis in der `HashMap` vorliegt. Oft werden mehrere Namen auf einmal (Darstellung einer Kontaktliste) angefragt und die entsprechenden Ressourcen sind noch nicht in die Modellverwaltung geladen worden. Diese Anfragen können durch die Verwendung von Threads parallel ausgeführt werden, um schneller zu einem Ergebnis zu gelangen.

7 Zusammenfassung und Ausblick

In dieser Arbeit wurde ein mobiler Social Semantic Web Client auf der Android-Plattform entworfen und entwickelt. Er besteht aus einer zweiteiligen Middleware, welche durch zwei Anwendungen genutzt wird. Das Softwaresystem erfüllt die Benutzeranforderungen weitestgehend, die am Anfang der Arbeit aufgestellt wurden (vergleiche Abschnitt 3). Es ist möglich die Web ID des Benutzers in das System zu laden und samt einer Kontaktliste anzuzeigen. Von dieser Kontaktliste ausgehen, kann man weitere Profile und dazugehörige Kontaktlisten laden und ansehen. Eine freie Eingabe ermöglicht es Profile anzuzeigen, die nicht über die Vernetzungsstruktur erreichbar sind. Neue Kontakte können zur Web ID des Benutzers hinzugefügt und der lokale Stand der Ressourcen Aktualisiert werden. Eine Integration in das Android-Adressbuch ist ebenfalls gelungen. Die Middleware des Social Semantic Web Clients ist über eine Schnittstelle für andere Programme auf der Plattform zugänglich. Damit ist es möglich geworden, auf einem mobilen Endgerät an dem Social Semantic Web teilzunehmen und nicht an verschiedene in sich geschlossene Netze gefesselt zu sein. Ab jetzt ist es jedem Teilnehmer des Social Semantic Webs möglich, unterwegs auf dem neusten Stand zu sein und seine Daten genau dort zu haben wo er sie braucht, in seinem mobilen Endgerät.

Bereits während des Entwurfs, aber auch während der Entwicklung, des Social Semantic Web Clients hat sich gezeigt, dass diese Kombination sozialer und semantischer Daten auf einer mobilen Plattform eine große Menge an Möglichkeiten eröffnet. Daher konnten in dieser Arbeit nicht alle aufbauenden Anwendungsfälle abgedeckt werden. Deshalb werden hier Anregungen für neue Anwendungen, die auf den Social Semantic Web Client aufbauen, beschrieben aber auch Stellen an denen der Client und die zugrundeliegende Middleware vervollständigt werden können.

7.1 Verbesserungsmöglichkeiten des Systems

Der Social Semantic Web Client konnte noch nicht seine volle Funktionalität entfalten. Hier werden einige Stellen aufgezeigt, an denen noch Ausbesserungs- und Weiterentwicklungsbedarf besteht.

Erweiterung der Authentifizierung mittels der Web ID Momentan ist die FOAF+SSL Authentifizierung (vergleiche Abschnitt 3.2.9) nur über eine HTTPS-Verbindung möglich. Viele Webservices nutzen allerdings Weiterleitungen von einer HTTP-Verbindung zu einer HTTPS-Verbindung. Diese Methode wurde bisher nicht berücksichtigt, kann aber in Zukunft ergänzt werden. Es ist auch nicht möglich ein Clientzertifikat durch den Social Semantic Web Cient selbst zu erzeugen (vergleiche Abschnitt 3.2.10).

Neue Freundschaftsrelationen schreiben Während der Arbeit hat sich herausgestellt, dass das Hinzufügen neuer Eigenschaften zu einer Web ID nicht ohne weiteres möglich ist. Aus dem URI der angefragten Ressource ergibt sich nicht immer direkt, auf welchem Weg neue Daten dieser Ressource hinzugefügt werden können. Es wäre möglich einer Web ID eine Eigenschaft hinzuzufügen, welche einen SPARQL/Update-URI (ähnlich `http://<OntoWiki-Installation>/sparql?query=<SPARQL-query>`) angibt. Über diesen können dann neue Tripel geschrieben werden. Momentan werden neue Freundschaftsrelationen in dem zu der Ressource zugehörigen Modell im Ordner `local` der Modellverwaltung (siehe Abschnitt 6.2.1) gespeichert.

Volltextsuche Die momentan implementierte Suche unterstützt nur die Eingabe einer vollständigen Web ID. Diese wird als einziges Ergebnis der Suche angezeigt. Dies genügt um beliebige Web IDs in das System zu laden (vergleiche Abschnitt 3.2.3) allerdings entspricht dies nicht den Erwartungen des Benutzers. Es existieren bereits einige Webservices (z. B. `http://foaf.qdos.com/find/` und `http://sindice.com/`), welche die Volltextsuche nach Web IDs ermöglichen. Es wäre möglich die Suche durch die Integration dieser Dienste zu erweitern.

Eine SPARQL-Schnittstelle hinzufügen Der `TripleProvider` besitzt einen Pfad `sparql/*`, welcher über die `query()`-Methode angefragt werde kann (siehe Abschnitt 5.2.2). Zum Zeitpunkt der Konzeption dieser Schnittstellen war jedoch noch keine SPARQL Unterstützung in Androjena vorhanden. Dies hat sich durch das ARQoid-Projekt[1] geändert, sodass dieser Pfad in Zukunft die Möglichkeit bieten soll SPARQL-Anfragen über die `ContentProvider`-Schnittstelle zu stellen. Der zweite Teil wird dazu durch eine URL-encodierte Zeichenkette ersetzt, welche die SPARQL-Anfrage beinhaltet.

Vereinigung der Middleware Die Designentscheidung die Middleware in eine Mobile Semantic Web Middleware und eine Mobile Social Semantic Web Middleware aufzuteilen hat sich während der Arbeit als nicht sehr vorteilhaft erwiesen. Um die Anwendung modular aufzubauen wurde sie in diese zwei Teile getrennt, doch hat sich gezeigt, dass

[1] Porting of Jena's ARQ to the Android platform: `http://code.google.com/p/androjena/wiki/ARQoid`

es nicht möglich ist, Objekte zwischen Prozessen auszutauschen. Hingegen können nur primitive Datentypen ausgetauscht werden. Dadurch geht der Komfort der Jena-Datenstruktur zur Verwaltung der Wissensbasis außerhalb der Mobile Semantic Web Middleware verloren.

Die Performance ist ein weiterer Aspekt. Auf jede Anfrage an die Mobile Social Semantic Web Middleware (`FoafProvider` oder `ContactProvider`) folgt eine Anfrage an die Mobile Semantic Web Middleware (`TripleProvider`). Durch die Einebnung der Struktur könnte auf diese Weise etwa die Hälfte der Anfragen gespart werden.

Möglich wäre eine einzige Middleware. Durch ein Pluginsystem wäre die Erweiterbarkeit durch spezielle Vokabularien (wie in diesem Fall FOAF, siehe Abschnitt 2.2) realisierbar.

7.2 Möglichkeiten aufbauender Applikationen

Die offenen `ContentProvider`-Schnittstellen `TipleProvider` und `FoafProvider` bieten die Möglichkeit viele aufbauende Anwendungen zu entwickeln. Dies können Social Semantic Web Anwendungen aber auch einfache nicht soziale Semantic Web Anwendungen sein. Denkbar sind z. B. eine mobile offline verfügbar DBpedia-Anwendung (Becker u. Bizer [2008]), Multiplayer-Spiele und Anwendungen zur Kommunikation und Darstellung von Beziehungen zwischen Personen.

Instant Messenger Roster Ein Instant Messenger kann die vorhandenen Daten nutzen, um eine Chat-Kontaktliste aufzubauen, mit welcher Kommunikation über die in den Profilen angegebenen Konten (Jabber/XMPP, ICQ, ...) angeboten werden kann. Ähnlich wie in „Instant Messenger Login and Roster“[2] beschrieben.

Statistikschnittstelle Eine Statistikschnittstelle wäre eine allgemeine Schnittstelle für Android-Anwendungen, um Statistische Daten der Web ID des Benutzers hinzuzufügen. Zum Beispiel Spielstände oder Trainingsergebnisse eines Sporttrackers.

[2] FOAF+SSL Use Cases: `http://esw.w3.org/Foaf%2Bssl/UseCases`

8 Quellcode der Software

Der Quellcode zu dem hier beschriebenen System wurde als OpenSource-Projekt in einem Google-Code-Projekt entwickelt und steht so auch zur Weiterentwicklung zur Verfügung. Die Google-Code-Projektseite ist unter `http://code.google.com/p/mssw/` erreichbar. Die aktuelle Entwicklungsversion des Quellcodes kann mittels Mercurial[1] heruntergeladen werden, näheres dazu findet man unter `http://code.google.com/p/mssw/source/checkout`.

Projektseite des mobilen Social Semantic Web Clients

Die beiden Pakete können außerdem über den Android-Market unter den Namen „Semantic Web Core“ und „FOAF/WebID Provider & Browser“ heruntergeladen werden (was bereits über 100 Nutzer getan haben).

Android Market Suche: „aksw“

[1] Mercurial Source Control Management: `http://mercurial.selenic.com/`

Literaturverzeichnis

[Arndt 2010] Arndt, Natanael: Entwicklung eines mobilen Social Semantic Web Clients. In: Fähnrich, Klaus-Peter (Hrsg.) ; Franczyk, Bogdan (Hrsg.) ; Gesellschaft für Informatik e.V. (Veranst.): *INFORMATIK 2010: Service Science – Neue Perspektiven für die Informatik* Bd. 2 Gesellschaft für Informatik e.V., 2010 (GI-Edition – Lecture Notes in Informatics), S. 1004–1005

[Becker u. Bizer 2008] Becker, Christian ; Bizer, Christian: PDF DocumentDBpedia Mobile: A Location-Enabled Linked Data Browser. In: *1st International Workshop about Linked Data on the Web*. Beijing, China, April 2008

[Berners-Lee 2009] Berners-Lee, Tim: Linked Data / W3C. Version: Juni 2009. `http://www.w3.org/DesignIssues/LinkedData.html`. 2009. – Forschungsbericht. – Design Issues

[Berners-Lee u. a. 1994] Berners-Lee, Tim ; Cailliau, Robert ; Luotonen, Ari ; Nielsen, Henrik F. ; Secret, Arthur: The World-Wide Web. In: *Commun. ACM* 37 (1994), Nr. 8, 76-82. `http://dx.doi.org/10.1145/179606.179671`. – DOI 10.1145/179606.179671. – ISSN 0001–0782

[Berners-Lee u. a. 2005] Berners-Lee, Tim ; Fielding, R. ; Masinter, L.: *Uniform Resource Identifier (URI): Generic Syntax*. Version: Januar 2005. `http://www.ietf.org/rfc/rfc3986.txt`

[Berners-Lee u. a. 2001] Berners-Lee, Tim ; Hendler, James ; Lassila, Ora: The Semantic Web. In: *Scientific American* 285 (2001), 17 Mai, Nr. 5, S. 34–44

[Brickley u. Miller 2004] Brickley, Dan ; Miller, Libby: FOAF Vocabulary Specification / FOAF Project. Version: August 2004. `http://xmlns.com/foaf/0.1/`. 2004. – Forschungsbericht. – Namespace Document

[Carroll u. a. 2004] Carroll, Jeremy J. ; Bizer, Christian ; Hayes, Patrick ; Stickler, Patrick: Named Graphs, Provenance and Trust / HP Laboratories Bristol. Version: 2004. `http://www.hpl.hp.com/techreports/2004/HPL-2004-57`. 2004 (HPL-2004-57). – Forschungsbericht

[Davis u. Jr 2010] Davis, Ian ; Jr, Eric V.: RELATIONSHIP: A vocabulary for describing relationships between people / vocab.org. Version: April 2010. `http://vocab.org/relationship/`. 2010. – Forschungsbericht. – Namespace Document

[Inkster u. a. 2010] Inkster, Toby ; Story, Henry ; Harbulot, Bruno ; Bachmann-Gmür, Reto: *WebID 1.0*. http://payswarm.com/webid/. Version: August 2010

[Lassila u. Swick 1999] Lassila, Ora ; Swick, Ralph R.: Resource Description Framework (RDF) Model and Syntax Specification / World Wide Web Consortium (W3C). Version: 22 Februar 1999. http://www.w3.org/TR/1999/REC-rdf-syntax-19990222/. 1999. – W3C Recommendation

[Meier 2010] Meier, Reto: *Professional Android 2 Application Development*. Indianapolis, Indiana : Wiley Publishing, Inc., 2010 (Wrox Programmer to Programmer). – ISBN 978–0–470–56552–0

[OMG 2005] OMG: *Unified Modeling Language Specification*. http://www.omg.org/spec/UML/ISO/19501/PDF/. Version: 1 2005

[Schandl 2010] Schandl, Bernhard: Replication and Versioning of Partial RDF Graphs. (2010). http://dx.doi.org/10.1007/978-3-642-13486-9_3

[Sommerville 2007] Sommerville, Ian: *Software Engineering*. 8. München : Pearson Studium, 2007

[Stickler 2005] Stickler, Patrick: CBD - Concise Bounded Description / W3C. Version: Juni 2005. http://www.w3.org/Submission/CBD/. 2005. – Forschungsbericht

[Story u. a. 2009] Story, Henry ; Harbulot, Bruno ; Jacobi, Ian ; Jones, Mike: FOAF+SSL: RESTful Authentication for the Social Web. In: Hausenblas, Michael (Hrsg.) ; Kärger, Philipp (Hrsg.) ; Olmedilla, Daniel (Hrsg.) ; Passant, Alexandre (Hrsg.) ; Polleres, Axel (Hrsg.): *Proceedings of the First Workshop on Trust and Privacy on the Social and Semantic Web (SPOT2009)*. Heraklion, Greece : CEUR-WS.org, Juni 2009

Abbildungsverzeichnis

1.1 Heutzutage muss sich jede Person in jedem sozialen Netzwerk ein Profil anlegen 5
1.2 Im Social Semantic Web sind Beziehungen zwischen Profilen auf unterschiedlichen Servern möglich 6

2.1 Darstellung eines RDF-Graphen mit Ressourcen, gerichteten und typisierten Kanten und einem Literal 10

3.1 Abfrage von Profilen (2) und Kontaktlisten (1 und 3) aus dem Social Semantic Web 14

5.1 Integration des Social Semantic Web Clients in das bestehende Android-System 23
5.2 Klassendiagramm der Mobile Semantic Web Middleware 25
5.3 Klassendiagramm des Mobile Social Semantic Web Middleware 30
5.4 Klassendiagramm FoafProvider 30
5.5 Klassendiagramm Browser 33
5.6 Klassendiagramm Adressbuchintegration 34
5.7 Nutzung des Adressbuchvokabular in einem Beispiel-RDF-Graphen . . . 36

6.1 Bildschirmfoto der Profilansicht 41
6.2 Bildschirmfoto der Kontaktliste 41
6.3 Bildschirmfoto der Suchfunktion 41
6.4 Bildschirmfoto der Accounttypenauswahl 43
6.5 Bildschirmfoto der Accountverwaltunge 43
6.6 Bildschirmfoto der Adressbuchsynchronisation 44
6.7 Bildschirmfoto des Adressbuchs 44
6.8 Bildschirmfoto der Kontaktansicht 45
6.9 Bildschirmfoto zeigt Kontakttyp (Web ID) 45

Listings

2.1 In dieser Arbeit verwendete RDF-Präfixe 10

5.1 Versionierung der Regeldatei `rules.n3` (Auszug) 29

6.1 Aufbau einer FOAF+SSL-Verbindung in Java 38
6.2 Beispiel einer Jena-Regel, Abbildung der Eigenschaft `foaf:name` auf das Adressbuchvokabular 40
6.3 Einbinden der Android-Suchfunktion in der Datei `AndroidManifest.xml` 42
6.4 Definition des Android-Kontotyps und der Synchronisationsschnittstelle in der Datei `AndroidManifest.xml` 42

Entwicklung eines mobilen Social Semantic Web Clients

Natanael Arndt

Zuerst veröffentlicht in: Fähnrich, Klaus-Peter (Hrsg.) ; Franczyk, Bogdan (Hrsg.) ; Gesellschaft für Informatik e.V. (Veranst.): *INFORMATIK 2010: Service Science – Neue Perspektiven für die Informatik* Bd. 2 Gesellschaft für Informatik e.V., 2010 (GI-Edition – Lecture Notes in Informatics)

© Gesellschaft für Informatik e.V.

Korrigierte Fassung.

Entwicklung eines mobilen Social Semantic Web Clients

Natanael Arndt

arndtn@gmx.de

Soziale Netzwerke spielen eine immer größer werdende Rolle in unserem täglichen Leben und die Einrichtung und Pflege unserer Accounts und Daten in diesen Netzwerken wird immer aufwändiger. Die meisten Sozialen Netzwerke sind in sich geschlossen und Nutzer können ihre Daten häufig nicht von einem Netzwerk zu einem anderen Netzwerk kopieren. Das uns bekannte, von Tim Berners-Lee beschriebene, World Wide Web (WWW) kennt diese Art künstlicher Abtrennung einzelner Teilnetze nicht:

> The idea of a boundless information world in which all items have a reference by which they can be retrieved [BLCL+94]

Das *Social Semantic Web* ist die Idee eines dezentralisierten und Service-unabhängigen Sozialen Netzwerkes, welches auf dem WWW und Technologien des Semantic Webs aufbaut. Das *Semantic Web* ist eine Erweiterung des WWW. Es bietet die Möglichkeit, mit Hilfe des Resource Description Frameworks (RDF, [LS99]), Ressourcen, d. h. Web-Dokumente aber auch nicht-virtuelle Ressourcen wie Personen und Gruppen, miteinander in Beziehung zu setzen. Das Friend of a Friend (FOAF) Projekt entwickelt ein RDF Vokabular [BM04], welches es ermöglicht auf einheitliche Weise Personen durch deren Namen, JabberID, Weblog, u. ä. zu beschreiben und deren persönliche Beziehungen untereinander darzustellen. Jeder Person wird dabei eine Web ID[1] zugeordnet, welche die Person eindeutig identifiziert. Werden diese Web IDs als Linked Data [BL09] publiziert, entsteht daraus ein dezentralisiertes Social Semantic Web.

Bisher existiert jedoch noch keine Möglichkeit in diesem Sozialen Netzwerk mit mobilen Endgeräten zu navigieren und daran teilzunehmen. Die Idee des mobilen *Social Semantic Web Clients* ist es daher, Teilnehmern dieses Sozialen Netzwerkes eine Möglichkeit zu geben, mit mobilen Geräten in diesem Netzwerk zu navigieren und es entsprechend zu manipulieren (z. B. neue Bekanntschaften dem eigenen Profil hinzuzufügen). Zu diesem Zweck wurde eine Infrastruktur aus Diensten entworfen und beispielhaft als `Content-Provider` für die Android-Plattform[2] implementiert.

Der *Mobile Semantic Web*-Dienst (`msw`) agiert dabei als Client innerhalb des Semantic Webs, welches er abfragt (z. B. indem er eine Ressource via Linked Data bezieht). Darüber hinaus stellt er seine Daten als RDF-Wissensbasis über eine einheitliche Schnittstelle auf der Plattform zur Verfügung. Zur Verwaltung der Wissensbasis auf der Android-Platt-

[1] Web ID: `http://esw.w3.org/WebID/`
[2] Android: `http://www.android.com/`

form wird Androjena[3] eingesetzt, welches eine Portierung des Jena Semantic Web Frameworks [McB02] ist. Auf die Schnittstellen des Mobile Semantic Web-Dienstes können beliebige Anwendungen zugreifen und dabei Ressourcen abfragen und z. B. angeben ob diese dauerhaft oder nur temporär gespeichert werden sollen. Der `ContentProvider` speichert die Daten aus dem Semantic Web als RDF-Tripel in einem Tripelstore.

Aufbauend auf dem Mobile Semantic Web-Dienst wurde ein weiterer Dienst entwickelt, welcher Aufgaben von Sozialen Netzwerken auf den bereitgestellten Daten durchführt (z. B. die Telefonnummer eines Bekannten erfragt). Dieser *Mobile Social Semantic Web*-Dienst (`mssw`) stellt wiederum eine Schnittstelle für Anwendungen zur Verfügung. So können diese Anwendungen das private Profil des Benutzers importieren und darin referenzierte WebIDs anderer Personen als Liste oder einzeln abrufen. Weiterhin besteht die Möglichkeit, über die Kontakte des eigenen Profils hinaus auf Daten der Freunde von Freunden zuzugreifen. Zu diesem Zweck können beliebige Ressourcen aus dem Social Semantic Web abgerufen werden.

Eine Evaluation der entwickelten Dienste wurde durch deren Verwendung in einer Android-Benutzeroberfläche durchgeführt. Diese Anwendung bietet die Möglichkeit, das eigene Profil mit einer Liste der Kontakte zu betrachten, über die Kontaktliste und eine freie Sucheingabe weitere Profile anderer Personen abzurufen, sowie neue Kontakte dem eigenen Profil hinzuzufügen. Darüber hinaus werden die direkten Kontakte des Benutzers in das Adressbuch des mobilen Systems integriert. Dadurch ergibt sich für den Benutzer des Sozialen Netzwerkes der Vorteil, dass er die ihm zur Verfügung gestellten Daten nicht in das System kopieren muss. Stattdessen werden die Daten seiner Bekannten aus dem Social Semantic Web heraus mit seinem Mobiltelefon synchronisiert. Dadurch ist jeder Teilnehmer des Sozialen Netzwerkes nicht nur kontinuierlich auf dem neusten Stand mit seinen Daten, er hat diese auch genau dort wo er sie braucht, in seinem mobilen Endgerät.

Literatur

[BL09] Tim Berners-Lee. Linked Data. Design Issues 18 Jun 2009, W3C, 2009. `http://www.w3.org/DesignIssues/LinkedData.html`.

[BLCL+94] Tim Berners-Lee, Robert Cailliau, Ari Luotonen, Henrik Frystyk Nielsen, and Arthur Secret. The World-Wide Web. *Commun. ACM*, 37(8):76–82, 1994.

[BM04] Dan Brickley and Libby Miller. FOAF Vocabulary Specification. Namespace Document 2 Sept 2004, FOAF Project, 2004. `http://xmlns.com/foaf/0.1/`.

[LS99] Ora Lassila and Ralph R. Swick. Resource Description Framework (RDF) Model and Syntax Specification. Technical report, 22 February 1999.

[McB02] Brian McBride. Jena: A Semantic Web Toolkit. *IEEE Internet Computing*, 6:55–59, 2002.

[3] androjena – Jena Android porting: `http://code.google.com/p/androjena/`

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 19. Oktober 2010 Unterschrift