Institut für Visualisierung und Interaktive Systeme
Universität Stuttgart
Universitätsstraße 38
D–70569 Stuttgart

Fachstudie Nr. 166

Visualisierung sozialer Beziehungen basierend auf Kontaktdaten

Vivian Eggert Huy Viet Le Kevin Wenz

Studiengang:	Softwaretechnik
Prüfer:	Prof. Dr. Albrecht Schmidt
Betreuer:	Dipl.-Inf. Niels Henze M. Sc. Alireza Sahami
begonnen am:	1. Juni 2012
beendet am:	1. Dezember 2012
CR-Klassifikation:	E.1

Kurzfassung

In der vorliegenden Arbeit wurde ein System entwickelt, welches die Beziehungen von Mobiltelefon Kontakten untereinander visualisiert. Das System besteht dabei aus drei Komponenten: Zum einen eine Android-Anwendung, mit der die Kontaktdaten auf den Smartphones ausgelesen und anschließend an einen Server gesendet werden. Dabei ist der Server für die Speicherung sowie Ermittlung von Beziehungen zwischen den Kontakten zuständig. Zum anderen wurde eine Web-Schnittstelle entwickelt, welche die abgespeicherten Kontakte samt Beziehungen in einem Graph visualisiert.

Abstract

The developed system visualizes relationships based on contact information of smartphones. This system consists of three components. The android application extracts contact information from a mobilphone and sends them to the server. In the server component all data are stored and relations between contact are determined. The web-client visualizes requested contacts and their relationships.

Inhaltsverzeichnis

1 Einleitung

Mobiltelefone sammeln und speichern vielfältige Information über Nutzer und deren Kontakte. Um diese Informationen verständlich darzustellen, wird eine angemessene Visualisierung benötigt. Dabei ist es wichtig, dass immer ein Überblick über alle Informationen gewährleistet ist und einzelne Kontakte sich nicht überdecken. Desweiteren müssen sie in ihrer Darstellung strukturiert sein und nach Zusammengehörigkeit gruppiert werden.
Diese Arbeit beschreibt ein System, welches sich mit dem Erheben der auf einem Telefon gespeicherten Daten und mit der Problematik der Darstellung von Kontakten und Beziehungen befasst.

1.1 Gliederung der Arbeit

Das System und somit die Arbeit wird in drei Komponenten gegliedert. Dieses System beinhaltet einen Android-Clienten, dessen Entwicklung in Kapitel 4 beschrieben wird. Dieser liest Kontakte aus Android Smartphones aus und überträgt diese an den Server. Der Server, welcher in Kapitel 3 beschrieben wird, ist für die Speicherung der Kontakte sowie die Ermittlung von Beziehungen zwischen diesen zuständig. Der Schwerpunkt dieser Arbeit liegt auf der graphischen Repräsentation der Beziehungen zwischen Kontakten, deshalb wurde der Web-Client entwickelt, dessen Aufbau in Kapitel 5 erläutert wird. Die Beziehungen der Kontakte werden dabei mithilfe eines Graphen visualisiert.

1.2 Zielsetzung

Das Ziel der Arbeit ist es ein System zu entwickeln, welches Nutzern erlaubt ihre Smartphone-Kontakte zu visualisieren. Der Schwerpunkt liegt dabei auf der Erkennung gemeinsamer Kontakte, welche Nutzer untereinander teilen. Diese sollen in der Visualisierung graphisch hervorgehoben werden. Zudem soll ein Nutzer den Kontaktgraphen anderer Nutzer sehen können, welche gemeinsame Kontakte mit dem Nutzer teilen. Diese Funktionalität soll durch einen Mausklick auf den Namen des gemeinsamen Kontakts aufrufbar sein.

Um Aspekte des Datenschutzes zu beachten, sollen die Telefonnummern aller Nutzer verschleiert werden. Lediglich die Namen, welche im Telefonbuch gespeichert sind, werden im Klartext übertragen und gespeichert. Der Nutzer soll seinen Kontaktgraphen nach dem Einloggen in das System sehen können.

2 Verwandte Arbeiten

Dieses Kapitel beschreibt bisher erfolgte Studien und Forschungsprojekte, welche thematisch mit dieser Arbeit verwandt sind. Es dient dazu einen Überblick über das Themengebiet zu erhalten und Grundlagen für diese Arbeit zu erfassen.

Das automatische Generieren von sozialen Netzwerken auf Basis von persönlichen Informationen wird in einigen Studien behandelt. Im Folgenden werden diese Studien vorgestellt und ihre Besonderheiten erläutert.
Van Alstyne et al. [VAZ03] generieren im Projekt EmailNet soziale Netzwerke, indem sie den Inhalt von E-Mails analysieren. Das Ziel dieser Arbeit ist es Datenerhebungen für Studien zu vereinfachen. Adamic und Adar [AA03] hingegen durchsuchen private, aber öffentliche Webseiten nach personenbezogenen Informationen. Diese Methode kann als Grundlage für das Analysieren von sozialen Beziehungen genutzt werden. Sie wurde bereits an verschiedenen Hochschulen getestet. Eine Arbeit von Culotta et al. [CBM04] an der University of Massachusetts-Amherst verbindet diese beiden Ansätze und beschreibt das Bauen eines sozialen Netzwerks durch Herausfiltern von Namen aus E-Mails. Diese Namen werden zum Auffinden von Homepages dieser Personen benutzt, welche wiederum nach Namen durchsucht werden können. Das Verfahren liefert in geeigneten Bereichen gute Ergebnisse und dient zum Beispiel dem Auffinden von Personen, die in ähnlichen Bereichen Arbeiten. Matsuo et al. schlagen den iterativen Prozess „Polyphonet“ [MMH+07] vor. Dieser sammelt ebenso personenbezogene Daten im Internet und nutzt zusätzlich Informationen aus anderen sozialen Netzen. Es wurden viele verschiedene Algorithmen implementiert und eine gute Skalierbarkeit gewährleistet. Während die meisten Projekte sich mit den Kontaktdaten beschäftigen setzten Schwartz et al. [SW93] auf einen andere Datengrundlage. Sie generieren ein soziales Netz auf der Basis gemeinsamer Interessen. Der generierte Graph umfasste bereits nach zwei Monaten ungefähr 50.000 Menschen.

Das Scannen von oftmals privaten Seiten im Internet ist aufgrund nicht gewährleisteter Privatsphäre nicht immer zu empfehlen, deshalb kann es oftmals besser sein ein soziales Netzwerk anhand der Kommunikationsdaten auf einem Mobiltelefon zu erstellen. In der Arbeit von Ankolekar et al. [ASLH09] werden mithilfe von „Friendlee“ auf Telefonier- und SMS-Interaktionen basierende soziale Netzwerke generiert. Roth et al. nutzt ebenfalls Interaktionbasierte (Anrufe, SMS, Chat, ...) Graphen [RBDD+10] und erweitert diesen Ansatz um einen Algorithmus der mögliche Freunde (`mutual friends`) vorschlägt. Xiang et al. [XNR10] erweitern diese Idee und schlagen eine Gewichtung anhand der Informationshäufigkeit, von der Beziehung zweier Knoten im Graphen vor. Diese Arbeit zeigt, dass das Gewichten der Kanten Verbesserungen der automatisierten Korreloration und Klassifizierung. Auch Gilbert et al. widmen sich in ihrer Arbeit „Predicting tie strength with social media“ [GK09] der

Gewichtung der Sträke einer Beziehung. Basierend auf verschiedenen sozialen Attributen wird versucht, diese Stärke vorherzusagen. Dabei wird eine Genauigkeit von 85% erreicht. Gemessen wird dieser Wert in „tie strength“, diese Einheit basiert auf einer Arbeit von Granovetter [Gra73]. Pietiläinen et al. erstellen in ihrer Arbeit „MobiClique“ [POL+09] ein Netzwerk, das mittels Bluetooth nach anderen Geräten sucht. Da Geräte direkt miteinander kommunizieren werden weder Server noch eine zentrale Datenspeicherung benötigt.

Das Durchsuchen des Internets und Bauen sozialer Kontaktgraphen bietet eine sehr gute Möglichkeit um abwechslungsreiche und echte Datensätze günstig zu generieren. Dabei ist ein korrekter ethnischer Umgang nahezu unmöglich. In dieser Arbeit sollen echte und auf realer Kommunikation basierende Daten verwendet und Kontaktdaten daher mithilfe von Mobiltelefonen gesammelt werden. Unser Hauptaugenmerk liegt dabei auf der Visualisierung der ermittelten Daten.

3 Server

In diesem Kapitel wird die Server-Komponente entwickelt, welche für die Speicherung der durch die Android-Anwendung ausgelesenen Daten sowie für die Generierung der Graphdatenstruktur zuständig ist. Mit der Graphdatenstruktur werden im Folgenden die Daten bezeichnet, welche vom Web-Client für die Visualisierung des Graphens zuständig sind. Der Aufbau dieser Graphdatenstruktur wird in Abschnitt 3.2 genauer erläutert.

Die Server-Komponente stellt zwei Schnittstellen zur Verfügung: Die eine Schnittstelle ist für den Empfang der von der Android-Anwendung gesendeten Daten zuständig, während die andere das Übertragen der Graphdatenstruktur an den Web-Client durchführt. Dabei wird die Server-Komponente auf einem Server ausgeführt, auf dem eine PHP-Installation in der Version 5.1[1] bereits vorinstalliert ist. Zudem ist eine Anbindung zu einer MySQL-Datenbank[2] über PHP verfügbar. Die PHP-Installation ist für das Ausführen der Server-Komponente zuständig, während in der MySQL-Datenbank die relevanten Daten zur Erstellung der Graphdatenstruktur gespeichert werden.

Bevor die funktionalen Anforderungen der Server-Komponente vorgestellt werden, wird zunächst auf das Protokoll eingegangen, welches für die Datenübertragung zwischen den einzelnen Schnittstellen verwendet wurde.

3.1 Protokoll

Die Kommunikation zwischen den einzelnen Komponenten wurde mithilfe eines auf JSON[3] basierenden Protokolls realisiert. Bei der Entwicklung des Protokolls wurde auf die Flexibilität geachtet, sodass die vorliegende Arbeit später erweitert werden kann.

Das Protokoll für die Android-Schnittstelle, sowie die Web-Client-Schnittstelle besteht aus jeweils zwei Containern. In diesen zwei Containern werden jeweils die Kontakte anhand ihrem `cid`-Wert und ihrem Namen gespeichert. Der zweite Container, welcher die Beziehungen der Kontakte untereinander speichert, ist nur an der Schnittstelle zum Web-Client nötig.

Im Folgenden wird der bereits erwähnte `cid`-Wert spezifiziert. Im Anschluss folgen zwei Ausschnitte der Protokolle, an denen das Protokoll nochmals genauer erläutert wird.

[1] PHP - Hypertext Preprocessor: http://www.php.net/
[2] MySQL-Datenbank: http://www.mysql.de/
[3] JSON: http://www.json.org

Listing 3.1 JSON-Protokoll Android-Client zu Server

```
{
        "uploader": {"cid": EINDEUTIGE_ID},
        "password": md5(PASSWORD),
        "contactList":[<CONTACT>, <CONTACT>, ...],
}

<CONTACT>:=
        {
                "name": CNAME1,
                "cid":["+4971112345","+491719876543",...],
                "type": <CONTACTLIST | SMS | CALL | EMAIL>,
                "weight":1
        }

Fehlercodes:
        401: invalid login
        200: request successful
```

3.1.1 cid-Wert

Der cid-Wert dient zur eindeutigen Identifikation eines Kontakts im zu entwickelnden System. Dabei ist einstellbar, wie dieser cid-Wert aufgebaut wird. In der Standardeinstellung entspricht der cid-Wert der Telefonnummer eines Kontakts. Dabei wird in der folgenden Arbeit angenommen, dass eine Telefonnummer die folgende Form besitzt:

```
+<Ländercode><Nummer>
```

Besitzt die Telefonnummer eine andere Form (beispielsweise eine Nummer aus dem Ausland), so wird versucht diese in das genannte Format zu bringen. In diesem Fall bedeutet dies, dass die führende Null in den deutschen Ländercode (+49) umgewandelt wird.

Zudem ist die Möglichkeit gegeben, Telefonnummern mithilfe eines MD5-Hash-Algorithmus'[4] zu verschlüsseln. Diese Option verhindert, dass Anwender fremde Kontakte anhand ihrer Telefonnummer sehen können. Dies dient dem Datenschutz.

3.1.2 Ausschnitte des Protokolls

In Listing 3.1 wird das Protokoll zwischen Android-Client und Server spezifiziert. Damit am Server bekannt wird, von welchem Anwender die Daten stammen, werden anhand der Attribute `uploader` und `password` ein Login am Server durchgeführt. Ist der Anwender anschließend bekannt, so können die Kontakte im Container `contactList` anschließend in die Datenbank eingetragen werden. Wie unterhalb des JSON-Abschnitts dargestellt, werden Kontakte anhand der Attribute name, cid, type und weight spezifiziert. Wurden ungültige

[4]MD5: http://www.tools.ietf.org/html/rfc1321

Listing 3.2 JSON-Protokoll Server zu Web-Client

```
{
        "uploader": {"cid":EINDEUTIGE_ID},
        "contacts": {"cid": EINDEUTIGE_ID},
        "relationships": [Relationship-Objekte]
}

Relationship-Objekt:
{
        "contact1": {"cid":EINDEUTIGE_ID},
        "contact2": {"cid":EINDEUTIGE_ID},
        "weight": 3391
}
```

Login-Daten übergeben, so wird der Fehlercode 401 zurückgegeben. Der Statuscode 200 steht für den erfolgreichen Abschluss der Operation.

Im Listing 3.2 ist das Protokoll abgedruckt, welches für die Datenübertragung von Server zu Web-Client zuständig ist. Dabei sind die Attribute uploader und die container `contacts` und `relationships` zuständig. Da die Beschreibung dieses Protokolls mit dem Aufbau der Graphdatenstruktur zusammenhängt, wird dieses in Abschnitt 3.2 beschrieben.

3.2 Funktionale Anforderungen

In diesem Abschnitt werden die funktionalen Anforderungen der Server-Komponente sowie die bereits erwähnten Schnittstellen genauer erläutert. Die Serverkomponente ist für die Speicherung, Erzeugung sowie die Übertragung der Graphdatenstruktur zuständig. Dazu werden die von der Android-Anwendung übertragenen JSON-Daten zunächst geparst und anschließend in die Datenbank geschrieben. Der Aufbau der Datenbank ist in Abschnitt 3.3 erläutert.

Werden vom Web-Client die Daten zur Erzeugung eines Graphens angefordert, so werden die in der Datenbank gespeicherten Daten zunächst ausgelesen und anschließend daraus die Graphdatenstruktur erstellt. Der Aufbau der Graphdatenstruktur wird in den folgenden Abschnitten erläutert. Zuletzt wird die Graphdatenstruktur, welche im JSON-Format repräsentiert werden, an den Web-Klienten übertragen.

Es folgen genauere Erläuterungen zu den Schnittstellen, der Datenbank sowie die Erzeugung der Graphdatenstruktur.

Schnittstelle Android Anhand dieser Schnittstelle empfängt die Serverkomponente Daten in Form eines JSON-Objekts von der Android Anwendung. Die übergebenen Daten werden in einer POST-Variable an die PHP-Datei übergeben. Die Attribute sowie der Inhalt wurde

in Kapitel 3.1 spezifiziert. Nach dem Parsen des JSON-Objekts erfolgt der in Kapitel 3.3.1 spezifierte Speichervorgang.

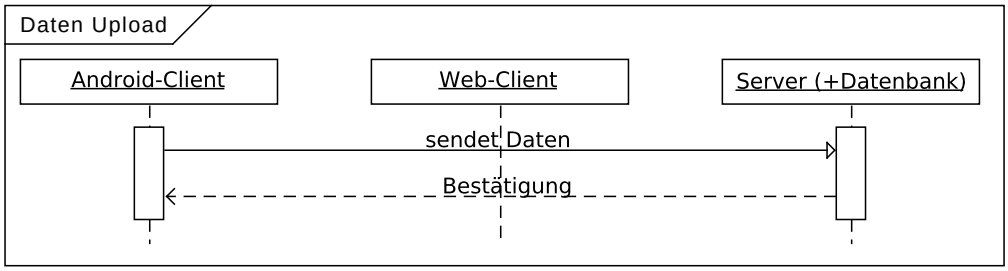

Abbildung 3.1: Sequenzdiagramm für Upload durch Android-Client

Schnittstelle Web-Client An den Web-Client, der für das Zeichnen des Graphens zuständig ist, sollen die Graphdaten verpackt in einem JSON-Objekt übergeben werden. Dazu sendet der Web-Client eine Anfrage, in der der Benutzername übergeben wird. Anhand des Benutzernamens wird ein entsprechender Graph aufgebaut und anhand von Knoten sowie Kanten in einem JSON-Objekt wieder zurückgesendet.

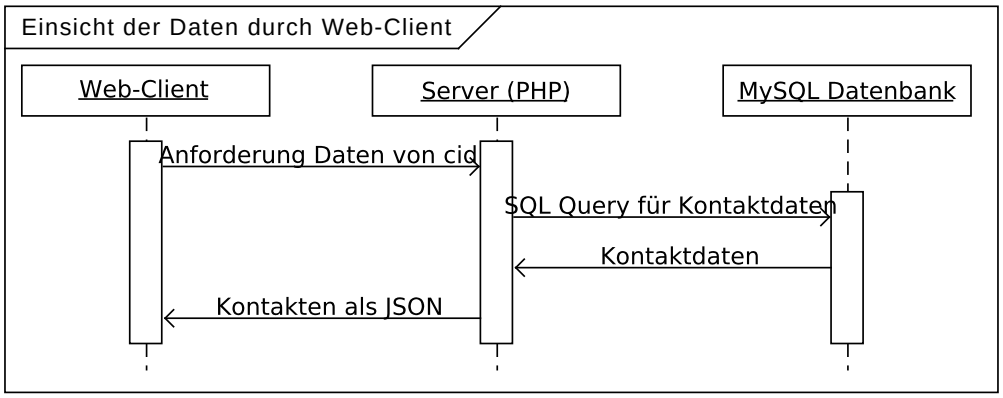

Abbildung 3.2: Sequenzdiagramm für Dateneinsicht durch Web-Client

Aufbau der Datenstruktur Das Format der Daten, welche an den Web-Client übertragen werden, wurde so konzipiert, sodass am Web-Client lediglich die Funktionalität zum Zeichnen des Graphens implementiert werden muss. Eine zusätzliche Verarbeitung des Graphens soll somit vermieden werden. Somit wird auch eine hohe Flexibilität gewährleistet, da der Aufbau des Graphens direkt am Server spezifiert werden kann und eine anschließende Änderung des Web-Clients nicht nötig ist.

Aus diesem Grund wurde der in Listing 3.2 dargestellte Aufbau gewählt. Der Aufbau beginnt in der obersten Ebene mit einem Container, welcher zwei weitere Container mit den Namen `contactList` und `relationships` besteht. Der Container `contactList` beinhaltet dabei alle

Kontakte, die sich in Kontaktliste des Benutzers befinden. Zu einem Kontakt werden dabei der `cid`-Wert, welcher einen Kontakt eindeutig identifiziert, sowie dem `name`-Wert. Anhand des `name`-Wertes können im Graphen die Namen der Kontakte angezeigt werden. Dabei sieht jeder Anwender die Kontakte mit den Namen, mit denen er die Kontakte in seiner Kontaktliste gespeichert hat.

Der Container `relationships` beinhaltet alle Kanten, welche die Kontakte im `contactList`-Container miteinander verbinden. Zu jeder Kante wird dabei der `cid`-Wert der zwei Kontakte gespeichert, welche miteinanader verbunden werden.

3.3 Aufbau der Datenbank

In diesem Abschnitt wird die Datenbank beschrieben, in der die Kontakt- und Beziehungsdaten gespeichert werden. Dabei werden im Folgenden die jeweiligen Tabellen der Datenbank aufgelistet und erläutert, welche Daten in dieser gespeichert werden. Für die Speicherung der Graphdatenstruktur werden drei Tabellen in einer relationalen Datenbank benötigt:

Tabelle Kontakte In dieser Tabelle werden Kontakte anhand ihrer ID gespeichert. Das Format der ID wird in Kapitel 3.1.1 erläutert. Zudem speichert hier der Web-Client, welcher der Kontakte bereits für unseren Dienst registriert ist. Für jeden registrierten Kontakt wird hier ebenfalls das gehashte Loginpasswort aufgeführt. Für die Graphdatenstruktur stellt diese Tabelle die Knoten zur Verfügung.

Tabelle Beziehungen Diese Tabelle spezifiziert die Beziehungen der bereits gespeicherten Kontakte. Dazu sind zwei Attribute notwendig, die jeweils einen Kontakt speichern. Anhand dieser Information sollen Instanzen dieser Tabelle Kanten der Graphdatenstruktur repräsentieren.

Tabelle Kontaktnamen Um für jeden Benutzer seine im Telefonbuch gespeicherten Namen im Graphen aufzuführen, werden hier die Namen aller Kontakte bezogen auf ihre Besitzer gespeichert.

3.3.1 Speicherung der Daten

Vom Android-Client empfangene Kontakte werden zunächst anhand ihrer ID in die Tabelle Kontakte eingetragen. Anschließend erfolgt der Aufbau von Beziehungen zwischen allen empfangenen Kontakten zu ihrem Besitzer. Dazu werden für alle empfangenen Kontakte ein Eintrag in der Tabelle Beziehungen angelegt, wobei die jeweiligen Attribute für den Anwender und für den Kontakt steht.

4 Android-Client

Das Android-Betriebsystem bildet weltweit einen der größten Anteile auf Mobiltelefonen [www11]. Dies und die freie Verfügbarkeit von Entwicklungstools sowie kostenlose Plattformen zur Verbreitung von Anwendungen qualifiziert es zum Durchführen von Studien und Entwickeln wissenschaftlicher Anwendungen.
Android-Anwendungen werden grundsätzlich in Java implementiert. Hinzu kommt XML für das Anordnen statischer Oberflächen. Anwendungen werden vom Nutzer manuell installiert. Die entwickelte Anwendung ist auf Geräten mit Android 2.1 und höher lauffähig.

4.1 Funktionale Anforderungen

Die Anwendung soll Kontakte und Telefonnummern des Adressbuchs auf dem Mobiltelefon auslesen. Diese werden an einen Server, dem definierten Protokoll (Abschnitt 3.1) entsprechend, gesendet. Um eine einfache Weiterentwicklung zu gewährleisten, soll außerdem auf eine flexible Anwendungsarchitektur geachtet werden. So soll es möglich sein, andere über Kommunikation aussagekräftige Daten auszulesen. Beispiele hierfür wären die Dauer der getätigte Anrufe oder die Häufikeit der Kommunikation per SMS, dabei soll auch die Stärke einer Beziehung (basierend auf der Dauer oder Häufigkeit) als Kantengewicht berücksichtig werden.

4.2 Implementierung

Der Android-Client besteht aus zwei Activity- und einigen Helferklassen (zum Beispiel: einer HTTP-Verbindungsimplementierung). Die beim Start erscheinende Activity implementiert die Abfrage der Nutzerdaten sowie der Serverinformationen. Nach Validierung dieser Daten - durch Überprüfung des Servers - wird der Nutzer zur zweiten Activity weitergeleitet. Dort werden die Kontaktdaten aus dem Mobiltelefon ausgelesen, aufbereitet und dem Protokoll entsprechend an den Server gesendet. Dabei werden Festnetz- und nicht deutsche Telefonnummern ausgefiltert. Alle deutschen Mobilfunknummern werden in das gewünschte Format (beschrieben in Abschnitt 3.1.1) konvertiert. Diese Activity implementiert das Interface *UploadActivity*, welches wiederum durch andere Datensammelarten (zum Beispiel: auf Anrufe basierend) implementiert werden kann.
HTTP-Verbindungen wurden als eigener Thread in eine Hilfsklasse ausgelagert, damit die Anwendung beim Auslesen nicht blockiert wird.
Bis auf Einstellungsdaten werden durch die Anwendung keine zusätzlichen Daten auf

Listing 4.1 Zugriff auf Telefoninhalt

```
// Zugriff auf Telefoninhalte mittels CONTENT_URI
Uri content =
       ContactsContract.CommonDataKinds.Phone.CONTENT_URI;
Cursor mqCur = managedQuery(content, // Uri für den Zugriff auf Telefonnummern
                         null,   // Spalten die zurückgegeben werden sollen
                         null,   // Welche Zeilen (SQL-WHERE)
                         null,   // Argumente der Zeilen Abfrage
                         null);  // Reihenfolge (SQL-ORDER BY)

// wähle Spalten für Name und Telefonnummer
while(mqCur.hasNext()){
  name = mqCur.getString(
    mqCur.getColumnIndex(ContactsContract.Contacts.DISPLAY_NAME)
  );
       number = mqCur.getString(mqCur.getColumnIndex(
              ContactsContract.CommonDataKinds.Phone.NUMBER)
       );
}
```

dem Mobiltelefon gespeichert. Um immer die aktuellsten Kontaktdaten benutzen zu können, werden diese mit jedem Ausführen erneut ausgelesen, verarbeitet und an den Server gesendet.

4.2.1 Auslesen von Kontaktdaten

Auf Mobiltelefonen[1] kann mit Hilfe der *content_uri* auf Telefoninhalte und so auch auf Kontakte zugegriffen werden. Die Inhalte können über einen Cursor, vergleichbar mit dem in Datenbaken verwendeten Cursor [PK05], abgerufen werden. Der Name eines Kontakts sowie dessen Telefonnummer müssen im entsprechenden Feld ausgewählt werden. Dieser Vorgang wird im Listing 4.1 beschrieben.

4.2.2 Zugriffsrechte

Damit eine Anwendung auf persönliche und sicherheitsrelevante Daten und Funktionen des Mobiltelefons zugreifen kann, müssen der Anwendung entsprechende Rechte zugewiesen werden. Im folgenden sind die durch diese Anwendung benötigten Rechte aufgeführt.

- READ_CONTACTS erlaubt der Anwendung auf die auf dem Mobiltelefon gespeicherten Kontakte zuzugreifen. In dieser Anwendung werden die Telefonnummern aus den Kontakten als Kontaktinformation an den Server gesendet.

[1] http://developer.android.com/reference/android/provider/ContactsContract.CommonDataKinds.Phone.html

- READ_PHONE_STATE erlaubt der Anwendung den Status des Mobiltelefons zu erfragen. Ein Beispiel dafür wäre ob gerade telefoniert wird. Das wird ebenfalls benötigt um auf die auf SIM-Karte gespeicherten Telefonnummer des eigenen Gerätes zuzugreifen.
- READ_SMS erlaubt der Anwendung gesendete und empfangene Nachrichten zu lesen. In manchen Fällen ist das Auslesen der SIM-Karte nicht möglich, dann wird Zugriffsrecht benötigt (Abschnitt 4.4).
- INTERNET erlaubt der Anwendung auf das Internet zuzugreifen. Dies ist notwendig um eigenen Kontaktdaten an einen zentralen Server zu Senden.

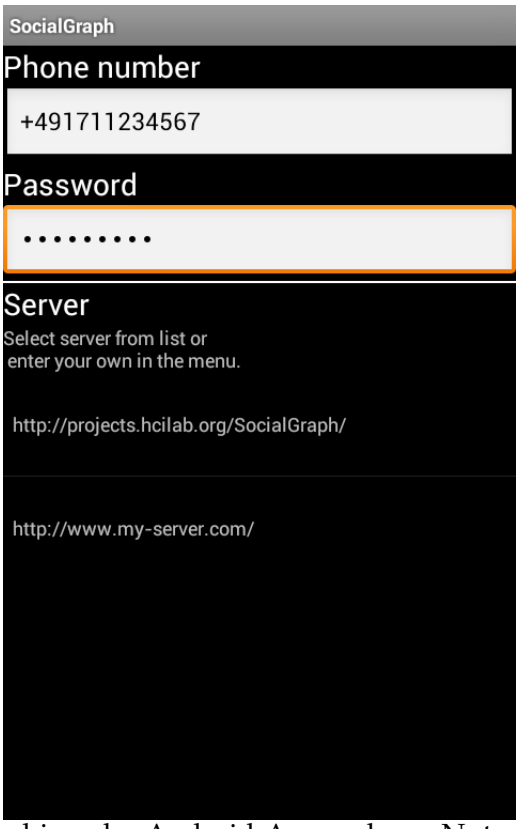

Abbildung 4.1: Start-Bildschirm der Android-Anwendung. Nutzername und Passwort können über die Textfelder eingegeben werden. Die Liste enthält Server, welche durch den Nutzer eingegeben und verändert werden können.

4.3 Design

Die Start-Activity bietet ein Formular zur Eingabe des Nutzernamens sowie das dazugehörige Passwort. Der Nutzername wird, wenn verfügbar, aus dem Mobiltelefon ausgelesen (Abschnitt 4.4). Das Formular enthält ebenfalls eine über das Activity-Menü editierbare Liste zur Auswahl eines Servers [4.1]. Die zweite Activity stellt bei erfolgreichem Senden der

Listing 4.2 Zugriff auf die Nummer eines Mobiltelefons

```
Context.getSystemService(Context.TELEPHONY_SERVICE).getLine1Number();
```

Daten die Antwort des Servers dar. Diese enthält eine Statusmeldung und die Anzahl der gespeicherten Kontakte.

4.4 Schwierigkeiten

Zur eindeutigen Identifikation eines Nutzers wird die Telefonnummer des Mobiltelefons verwendet (in 3.1.1). Android bietet mit der Klasse *TelephonyManager* die Möglichkeit auf Telefonnummer und SMS zuzugreifen. Der Zugriff auf die Telefonnummer wird im Listing 4.2 beschrieben. Auf einigen Geräten liefert diese Methode einen leeren String zurück. Ein möglicher, aber auch nicht sicher zum Erfolg führender Ansatz wäre es eingehende SMS auszulesen und dort den Empfänger zu speichern. Es ist auch nicht klar, was geschehen soll, wenn ein Telefon zwei Nummern besitzt (Dual-SIM).
Für die geforderten Anforderungen bieten Java und Android bereits geeignete Lösungen. So konnte für die HTTP-Verbindung das Java-Paket *http* verwendet werden. Das JSON-Objekt zur Übertragung (siehe 3.1) der Kontaktdaten wurde als String zusammen gebaut, ohne eine JSON-Klassen zu verwenden. Einzig das Filtern von Telefonnummern - wenn mehr als eine Nummer pro Kontakt verfügbar ist - stellt eine Herausforderung dar. Die wenigen deutschen Mobiltelefon-Vorwahlen ließen sich mit den von Java bereitgestellten Regulären-Ausdrücken auswählen.
Es musste auf keinerlei fremde Bibliotheken zurückgegriffen werden.

5 Web-Client

In diesem Kapitel wird die Web-Client-Komponente beschrieben. Sie dient als Schnittstelle zur Datenbank und dem Visialisieren des Kontaktgraphens. Desweiteren dient er als Anmeldemaske für registrierte Benutzer, sowie zur Registration neuer Benutzer.

5.1 Funktionale Anforderungen

Der Web-Client soll als Schnittstelle zu dem Server und der Visualisierung dienen. Eine Registration und Authentifikation für die Datenbank sollen von dem Web-Client aus erreichbar sein. Desweiteren soll der Android-Client von der Hauptseite heruntergeladen werden können. Nachdem sich der Benutzer angemeldet hat, wird sein persönlicher Kontaktgraph erzeugt. Hat ein Nutzer Kontakte in seinem Kontaktgraph, die ebenfalls in dem System registriert sind, kann der Nutzer zu dessen Kontaktgraph wechseln um so herauszufinden ob der gesuchte Kontakt dort vorhanden ist. Der Web-Client enthält auf der Startseite eine kurze Erläuterung des Dienstes. Der Nutzer kann sich auf der Hauptseite für den Dienst registrieren und den Android-Client herunterladen. Nachdem er Daten von seinem Android-Client auf den Server hochgeladen hat, kann er sich über den Web-Client einloggen. Hat der Nutzer sich erfolgreich eingeloggt, wird sein persönlicher Kontaktgraph erzeugt. Wenn ein persönlicher Kontakt des Nutzers auch dieses System nutzt, kann der Nutzer durch anklicken des Kontaktes, auf dessen Kontaktgraph gelangen. Aufgrund einiger Aspekte des Datenschutzes wurde das System so konstruiert, dass die Telefonnummern der Kontakte nicht direkt eingesehen werden können.

Listing 5.1 Pseudocode Erstellung des Graphen

```
json = RECEIVE_JSON
contactList = json.contactList
relationshipList =json.relationshipList
// generate all contacts and set an edge between owner and his contacts
FOR EACH contact IN contactList
        node = new Node
        node.color = 'orange'
        node.clickable = false
        IF contact == json.owner THEN node.color = 'green'
        graph.addNode(node)
        IF contact != json.owner THEN
                graph.addEdge(from:json.owner,node,'black')
        FI
END FOR EACH
// generate relationships between the owner contacts
FOR EACH contact IN relationshipList
        graph.addEdge(contact.cid1,contact.cid2,'blue')
        node1 = graph.getNode(contact.cid1)
        node2 = graph.getNode(contact.cid2)
        node1.color = 'red'
        // clickable=true because he is the uploader of all relationship information
            of this contact
        node1.clickable = true
        node2.color = 'red'
END FOR EACH
```

5.2 Algorithmus

Der in Listing 5.1 beschriebene Algorithmus empfängt zwei Listen. Die erste Liste beinhaltet die eigenen Kontakte, während die zweite Liste die Kontaktbeziehungen zwischen den Kontakten der ersten Liste beinhaltet, falls diese überhaupt Kontaktbeziehungen besitzen. Die Kontaktlisten werden im JSON-Format übertragen. Es wird zunächst über die eigene Kontaktliste iteriert, wobei für jeden Kontakt ein Knoten generiert wird, zwischen dem Benutzer und seinen Kontakten werden Kanten hinzugefügt, danach wird der Benutzerknoten farblich markiert. Jetzt wird die Liste der Kontaktbeziehungen durchlaufen, von jedem Kontakt in dieser Liste wird überprüft ob er eine Beziehung mit einem schon existierenden Knoten hat. Wenn eine Übereinstimung gefunden wurde, wird eine farbige Kante zwischen den beiden Kontakten erzeugt. Diese Kante soll darstellen, dass sich beide Kontakte untereinander kennen. Desweiteren wir der ausgehende Kontaktknoten farblich markiert um so schneller Kontaktgruppen im Kontaktgraph identifizieren zu können.

5.3 Design

Die Hauptseite wurde mit dem Front-End Framework Bootstrap[1] in Verbindung mit der Javascript-Bibliothek JQuery[2] erstellt. Sie stellt ein Registrierungsformular, sowie die Anmeldemaske für registrierte Benutzer zu Verfügung. Ein QR-Code, welcher die URL zum Download der Android-Anwendung beinhaltet, wird auf der Startseite eingeblendet.

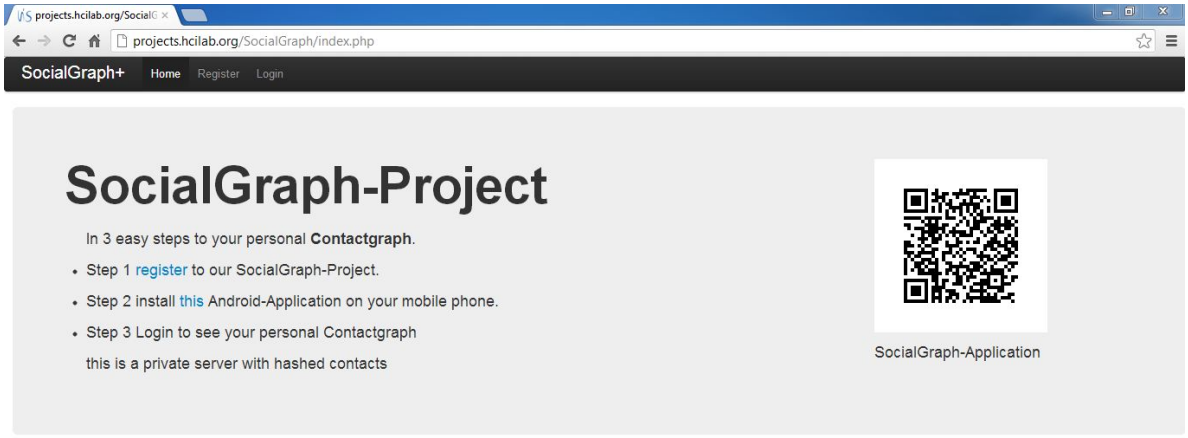

Abbildung 5.1: Startseite des Web-Clients

Die erste ausführbare Version verwendete die Javascript-Bibliothek *Moowheel*[3] zur Erstellung des Kontaktgraphens. *Moowheel* ordnet alle Kontakte in einem Kreis an und hebt die Beziehungen mit verschieden farblichen Kanten hervor. Die Vorteile dieser Bibliothek war die geringen Systemanforderungen an den Web-Browser. Der Graph wurde performant gezeichnet und unterstützte keine Bewegungsanimation der Knoten, somit war der Zeichenaufwand minimal. Ein weiterer Vorteil war die Tatsache, dass der Graph eine große Anzahl an Kontaktknoten ohne Überlappungen oder Platzprobleme darstellen konnte. Die Graphvisualisierung hatte aber auch Nachteile. Es war in dieser Graphvisualisierung schwer erkennbar, welche Beziehungen die Kontakte untereinander haben. Auch die Kantenfarben konnten nicht explizit gewählt werden, es konnte lediglich ein Farbschema gewählt werden. Deswegen wurde auf die Verwendung dieser Bibliothek verzichtet und auf die im folgenden Abschnitt beschriebene Bibliothek zurückgegriffen.

[1]Bootstrap: http://twitter.github.com/bootstrap/
[2]JQuery: http://jquery.com/
[3]Moowheel: http://labs.unwieldy.net/moowheel/

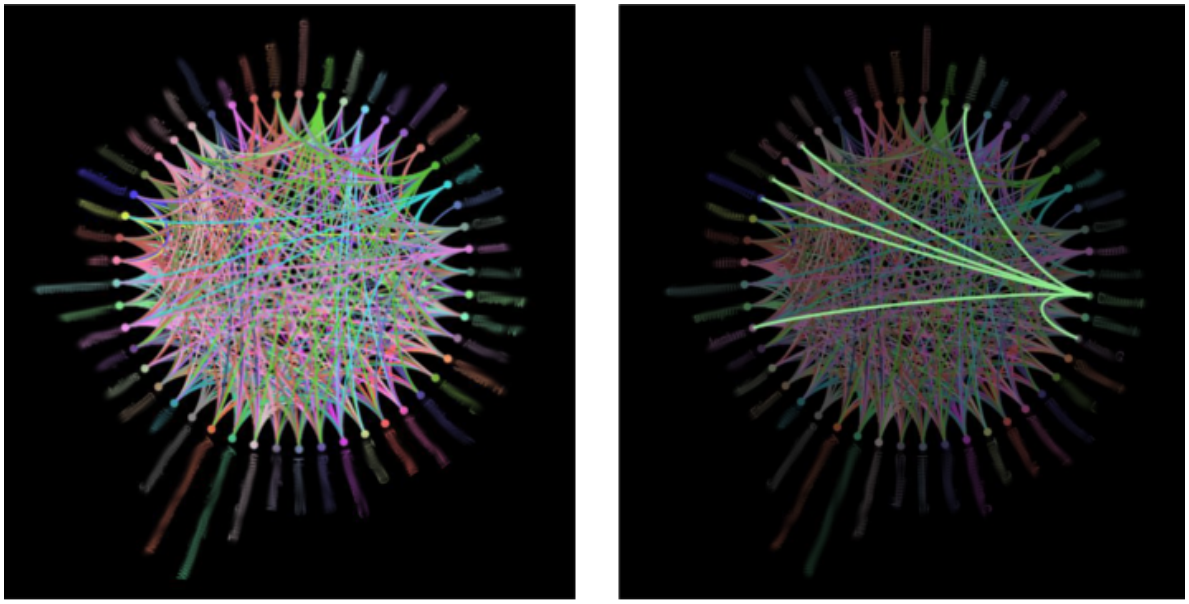

Abbildung 5.2: Links: Moowheel-Bibliothek Darstellung eines Graph mit Kontakten
Rechts: Moowheel-Bibliothek Darstellung einse Graph mit ausgewähltem Kontakt

Der folgende Abschnitt beschreibt warum die Entscheidung auf die Javascript-Bibliothek *Arbor*[4] gefallen ist. Das System verwendet *Arbor*, da die Kontakte als Knoten und die Beziehungen zwischen den Kontakten als Kanten in einem zweidimensionalen Raum strukturiert visualisiert werden können. Je mehr Kanten ein Knoten hat, desto mehr wird er von den verbundenen Knoten angezogen. So enstehen gruppierte Anordnungen genau dort, wo sich die Kontakte untereinander kennen. Desweiteren werden Gruppierungen unter Kontakten durch gleichfarbige Knoten hervorgehoben. Der Benutzer steht dabei immer im Zentrum des Graphen und wird nicht von den anderen Kontakten beeinflusst. So wird eine schnell ersichtliche Übersicht über die einzelnen Gruppen erzeugt. *Arbor* bietet eine Vielzahl von Einstellungsmöglichkeiten der Graphdarstellung an.

- repulsion - Die Kraft mit der sich die Knoten untereinander abstoßen. Standardwert ist 1000.
- stiffness - Die Starre der Kanten. Standardwert ist 600.
- friction - Die Dämpfungen der Bewegungen. Standardwert ist 0.5.
- gravity - Optionale Kraft, die alle Knoten zum Mittelpunkt zieht. Standardwert ist false.
- fps - Bildwiederholungsrate. Standardwert ist 55.
- dt - Zeitintervall für Animationsausführung. Standardwert ist 0.02.

[4]arbor: http://arborjs.org/

- precision - Genauigkeit der Berechnungen. Standardwert ist 0.6.

Im Folgenden werden die gewählten Parameter für die genannten Einstellungen erläutert. Diese entstanden durch Erfahrungswerte, die eine optimale Visualisierung für das gegebene System anstreben.

- repulsion = 200
  Wenn der Graph viele Kontakte beinhaltet, würde eine zu große Abstoßung dazu führen, dass sich die Knoten gegenseitig wieder zu dicht an andere Knoten hinstoßen würde. Das erzeugt zuviel Spannung im Graph, deswegen wurde ein niedriger Wert gewählt.
- stiffness = 250
  Mit einem niedrigeren Wert können sich die Knoten freier bewegen.
- friction = 0
  Der Standardwert hat die Bewegung des Graphen verlangsamt und wurde deswegen komplett ausgeschalten.
- gravity = true
  Da sich der Nutzer im Mittelpunkt befindet und sich alle seine Kontakte um ihn anordnen sollen wurde der Wert auf true gesetzt Standardwert ist false.
- fps = 55
  Die Bildwiederholungsrate wurde auf dem Standardwert gelassen.
- dt = 0.02
  Der Zeitintervall für Animationsausführung wurde ein wenig verkürzt um eine flüßigere Darstellung zu gewährleisten.
- precision = 0.6
  Der Standardwert wurde als Kompromiss zwischen Genauigkeit und Rechenaufwand beibehalten.

Die Nutzung der Bibliothek *Arbor* ist im Listing 5.1 beschrieben.

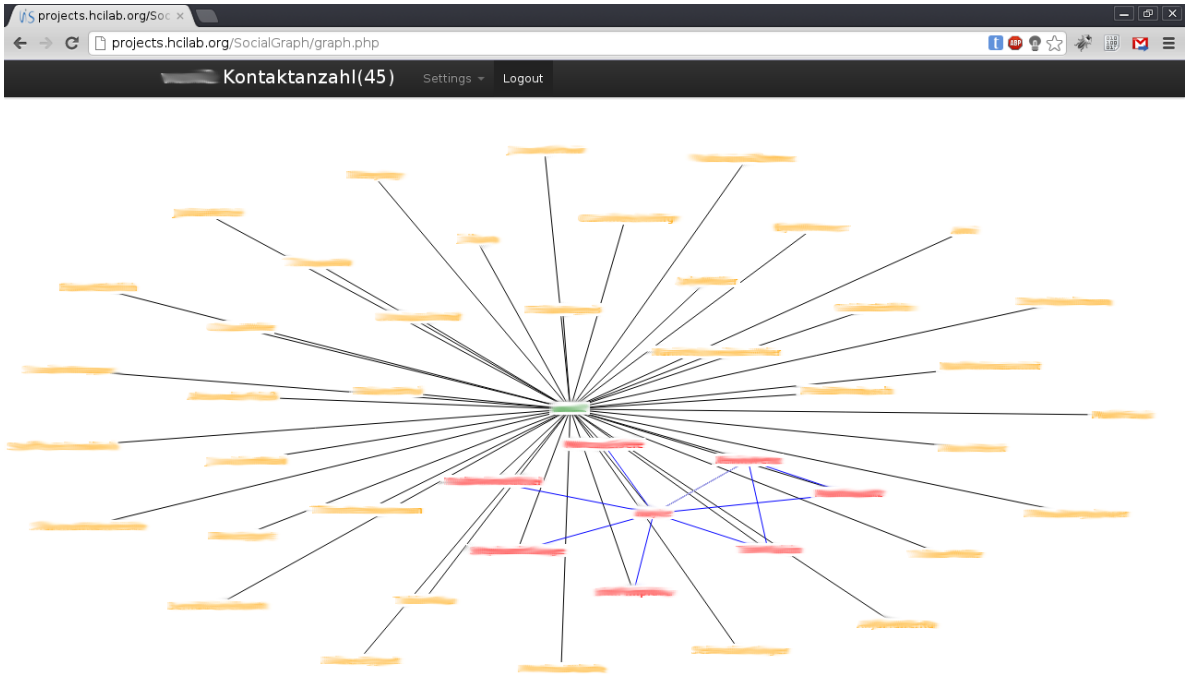

Abbildung 5.3: Endgültige Version des Webclients mit geladenem Graphen. Die roten Kontakte stellen eine Kontaktgruppe dar, die sich untereinander kennen. Die Grafik soll die dynamik des Graphen verdeutlichen. Die roten Kontakte haben sich zu einer Gruppe zusammengezogen. Die Verbindungsstärke hängt von der Anzahl der Kanten ab, die sie mit anderen Kontakten teilen. Die Namen wurden aus Datenschutzgründen unkenntlich gemacht.

5.4 Schwierigkeiten

Bei der Verwendung von *Arbor* muss beachtet werden, dass diese Bibliothek sehr hardwarelastig ist und auf leistungsschwächeren Geräten keine flüssige Animation des Graphen möglich ist. Um die Dynamik des Graphen zu gewährleisten und dem Benutzer die Möglichkeit zu geben Kontakte manuell zu verschieben und neu anzuordnen, muss sich der Graph in definierten Abständen neu zeichnen. Somit hat man eine kontinuierliche Auslastung des Systems.

6 Zusammenfassung

In der vorliegenden Arbeit wurde ein System entwickelt, welches die Beziehungen von Mobiltelefon-Kontakten untereinander visualisiert. Dabei wurde durch die Android-Anwendung eine Möglichkeit entwickelt, mit der registrierte Anwender ihre Kontaktliste an das System übertragen konnten. Aus diesen Kontakten wurde anschließend durch die Server-Komponente eine Graphdatenstruktur erstellt, welche letztendlich vom Web-Client anhand eines Kontaktgraphen visualisiert wird.

Anhand diesem Kontaktgraphen können Anwender des Systems in Verbindung zu neuen Kontakten treten, indem diese durch die Kontaktlisten anderer registrierter Anwender navigieren. Dies geschieht durch einen Klick auf den Namen eines registrierten Anwenders, was zur Anzeige des Kontaktgraphen des angeklickten Anwenders führt.

6.1 Fazit

Das entwickelte System visualisiert Beziehungen zwischen den Kontakten von registrierten Anwendern. Dies geschieht durch die Graphbibliothek *Arbor*. Bevor die Entscheidung für die genannte Graphbibliothek fiel, wurden davor bereits die Bibliotheken *GraphDracula* und *MooWheel* eingesetzt und evaluiert. *GraphDracula* war nur für einen schnellen Prototyp geeignet, da der Funktionsumfang dieser Bibiliothek für unser System nicht ausreichend war. Aufgrund der Unübersichtlichkeit in *MooWheel* und der fehlenden Ästhetik, fiel die Entscheidung letztendlich auf *Arbor*. Auch die Möglichkeit zur Zentrierung des Anwenders sowie zahlreiche Einstellungen, welche in den zwei bereits eingesetzten Bibliotheken nicht vorhanden war, bekräftigte das Argument zur Verwendung von *Arbor*. Trotz der Vorteile sind auch in *Arbor* einige Nachteile zu finden. So stellte sich nach einigen Testdurchläufen heraus, dass die Performanz signifikant sinkt, wenn die darzustellende Kontaktliste aus mehr als 300 Kontakten bestand.

Für die Speicherung und eindeutigen Identifizierung von Kontakten wurde vor der finalen Version die Telefonnummern verwendet. Aus Datenschutzgründen wurde diese Variante überarbeitet und Telefonnummern durch ihre jeweiligen Hashwerte ersetzt.

6.2 Ausblick

Das System visualisiert in der vorliegenden Arbeit die Beziehungen der Kontakte untereinander, wobei die Kanten lediglich durch den Inhalt der Kontaktlisten bestimmt werden.

Um Kanten zu gewichten und somit die Stärke der Beziehungen der Kontakte untereinander zu modellieren, können zudem die Anzahl an ausgetauschten SMS-Nachrichten oder getätigten Anrufen hinzugenommen werden. Der aktuelle Stand des Systems bietet dazu das Attribut *weight* in der Datenbank an, das noch nicht durch das System betrachtet wird. Durch Berechnungen von Häufigkeit, Dauer und Uhrzeiten der Telefonate unter Einfluss von Techniken des maschinellen Lernens können zudem detailliertere Angaben erstellt werden, welche als Kantengewichte verwendet werden. Durch die Vielseitigkeit von *Arbor* können auch statische Graphen erzeugt werden um Rechenleistung, für Graphen mit sehr vielen Knoten, zu sparen.

Literaturverzeichnis

[AA03] L. A. Adamic, E. Adar. Friends and neighbors on the Web. *Social Networks*, 25(3):211 – 230, 2003. doi:10.1016/S0378-8733(03)00009-1. URL http://www.sciencedirect.com/science/article/pii/S0378873303000091. (Zitiert auf Seite 9)

[ASLH09] A. Ankolekar, G. Szabó, Y. Luon, B. A. Huberman. Friendlee: A Mobile Application for Your Social Life. In *MobiCASE*, S. 331–334. 2009. (Zitiert auf Seite 9)

[CBM04] A. Culotta, R. Bekkerman, A. McCallum. Extracting social networks and contact information from email and the Web. In *CEAS*. 2004. (Zitiert auf Seite 9)

[GK09] E. Gilbert, K. Karahalios. Predicting tie strength with social media. In *CHI*, S. 211–220. 2009. (Zitiert auf Seite 9)

[Gra73] M. S. Granovetter. The Strength of Weak Ties. *American Journal of Sociology*, 78(6):pp. 1360–1380, 1973. URL http://www.jstor.org/stable/2776392. (Zitiert auf Seite 10)

[MMH+07] Y. Matsuo, J. Mori, M. Hamasaki, T. Nishimura, H. Takeda, K. Hasida, M. Ishizuka. POLYPHONET: An advanced social network extraction system from the Web. *J. Web Sem.*, 5(4):262–278, 2007. (Zitiert auf Seite 9)

[PK05] J. Pipes, M. Kruckenberg. *Pro MySQL*. Springer-Verlag, 2005. (Zitiert auf Seite 18)

[POL+09] A. K. Pietiläinen, E. Oliver, J. LeBrun, G. Varghese, C. Diot. MobiClique: middleware for mobile social networking. In *WOSN*, S. 49–54. 2009. (Zitiert auf Seite 10)

[RBDD+10] M. Roth, A. Ben-David, D. Deutscher, G. Flysher, I. Horn, A. Leichtberg, N. Leiser, Y. Matias, R. Merom. Suggesting friends using the implicit social graph. In *KDD*, S. 233–242. 2010. (Zitiert auf Seite 9)

[SW93] M. F. Schwartz, D. C. M. Wood. Discovering Shared Interests Using Graph Analysis. *Commun. ACM*, 36(8):78–89, 1993. (Zitiert auf Seite 9)

[VAZ03] M. Van Alstyne, J. Zhang. Emailnet: A system for automatically mining social networks from organizational email communication. *Ann Arbor*, 1001:48109, 2003. (Zitiert auf Seite 9)

[www11] www.gartner.com. Gartner Says Android to Command Nearly Half of Worldwide Smartphone Operating System Market by Year-End 2012, 2011. URL http://www.gartner.com/it/page.jsp?id=1622614. (Zitiert auf Seite 17)

[XNR10] R. Xiang, J. Neville, M. Rogati. Modeling relationship strength in online social networks. In *WWW*, S. 981–990. 2010. (Zitiert auf Seite 9)

Alle URLs wurden zuletzt am 30. 10. 2012 geprüft.

Erklärung

Hiermit versichere ich, diese Arbeit selbständig verfasst und nur die angegebenen Quellen benutzt zu haben.

(Vivian Eggert Huy Viet Le Kevin Wenz)