

# Unifying Social Networks for Smart Phones

Jose Luis de la Peña



Kongens Lyngby 2012  
IMM-M.Sc.-2012-51

Technical University of Denmark  
Informatics and Mathematical Modelling  
Building 321, DK-2800 Kongens Lyngby, Denmark  
Phone +45 45253351, Fax +45 45882673  
[reception@imm.dtu.dk](mailto:reception@imm.dtu.dk)  
[www.imm.dtu.dk](http://www.imm.dtu.dk) IMM-M.Sc.-2012-51

# Abstract

---

Smartphones and Social networks are two concepts closely related which are in an evolution process and upswing. The amount of information and possibilities available is growing fast and it is becoming increasingly harder to manage and to understand the resources at our disposal. This thesis aims to provide a tool to simplify the process of comprising the social networks included in Facebook, Twitter and LinkedIn. The tool developed is a system that has been called Social Unifier, which contains three main networking functionalities. First of all, Social Unifier is built over a NoSQL database system called Borges that presents a solution to manage and access to all the social information of the users in a fast, reliable way. Besides that, the system has an algorithm that finds matching profiles for those users connections across the three social networking sites. Ultimately, the system also finds suggested connections for the users by taking advantage of the information that is already stored in the system. These three functionalities are operated from an Android device application which provides the user interface for the whole Social Unifier system. The system has been tested by DTU students and the results of those tests have been detailed and analyzed here. This thesis relates and discusses the evolution of this research and the possibilities it may offer in the future.



# Preface

---

This thesis was prepared at the department of Informatics and Mathematical Modelling at the Technical University of Denmark in fulfilment of the requirements for acquiring an M.Sc. in Telecommunications.

The thesis is the result of work carried out in the period from November 2011 to June 2012 with a workload of 35 ECTS points, under the supervision of Jakob Eg Larsen and Sune Lehmann Jørgensen.

Lyngby, 01-June-2012

Jose Luis de la Peña



# Acknowledgements

---

First and foremost, I would like to thank my supervisors, Jakob Eg Larsen and Sune Lehmann Jørgensen, for their advice and patience, and especially for giving me the opportunity of working on this beautiful project.

I also need to thank the people who shared their lives with me in Denmark. Felix Rubio, who is my family here, and who always goes through what I go and the members of "Grandes", who made me love Copenhagen as much as I do.

Last and most important, I want to thank my family, my aunt and my uncle Nani Fernández and Javier Parra, my grandmother "Pepita" Mira, my parents Jose Luis and María Jesús, and my brothers, Óscar and Daniel. who made me who I am, and always believed in me and supported me. They can not imagine how grateful I am.





# Contents

---

<b>Abstract</b>	<b>i</b>
<b>Preface</b>	<b>iii</b>
<b>Acknowledgements</b>	<b>v</b>
<b>1 Introduction</b>	<b>1</b>
1.1 Problem description . . . . .	1
1.2 Thesis aim . . . . .	2
1.3 Methodology . . . . .	2
1.3.1 Resources . . . . .	3
1.4 Outline . . . . .	3
<b>2 First Approach</b>	<b>5</b>
2.1 Python . . . . .	5
2.2 Twitter . . . . .	6
2.3 Facebook . . . . .	6
2.4 LinkedIn . . . . .	7
2.5 CouchBase . . . . .	7
2.6 First approach conclusions . . . . .	7
<b>3 Borges Database</b>	<b>9</b>
3.1 CouchBase . . . . .	9
3.1.1 Queries, JSON Documents and HTTP requests . . . . .	9
3.1.2 Futon . . . . .	10
3.2 Design . . . . .	10
3.2.1 Users . . . . .	10
3.2.2 Connections . . . . .	11
3.2.3 Social Unifier Information . . . . .	13

3.2.4	Graph . . . . .	13
<b>4</b>	<b>Analysis and Design</b>	<b>15</b>
4.1	System Overview . . . . .	15
4.1.1	Components . . . . .	16
4.1.2	Communication between Components . . . . .	17
4.2	Mobile Device Application . . . . .	17
4.2.1	UML Use Case Diagram . . . . .	18
4.2.2	UML Sequence Diagram . . . . .	19
4.2.3	User interface . . . . .	21
4.3	Summary . . . . .	21
<b>5</b>	<b>Implementation</b>	<b>23</b>
5.1	Overview . . . . .	23
5.1.1	Eclipse Project . . . . .	24
5.2	Interface . . . . .	24
5.3	Social Networks . . . . .	25
5.3.1	Facebook . . . . .	26
5.3.2	Twitter . . . . .	26
5.3.3	LinkedIn . . . . .	27
5.4	Connection to the Server . . . . .	27
5.4.1	Database Class . . . . .	27
5.4.2	Upload Class . . . . .	28
5.5	Check Matches . . . . .	28
5.6	Graph View . . . . .	30
5.6.1	JavaScript . . . . .	30
5.6.2	Graph View Discussion . . . . .	30
5.7	Matching Script . . . . .	31
<b>6</b>	<b>Test and Results</b>	<b>33</b>
6.1	Test Plan . . . . .	33
6.2	Matching Results . . . . .	34
6.3	Network . . . . .	34
<b>7</b>	<b>Discussion</b>	<b>39</b>
7.1	Results Analysis . . . . .	39
7.2	Development Stage of the System . . . . .	40
7.3	Further Work . . . . .	41
<b>8</b>	<b>Conclusion</b>	<b>43</b>
<b>A</b>	<b>Screen Captures</b>	<b>45</b>

<b>B</b>	<b>Class Overview</b>	<b>53</b>
B.1	Social Unifier Activities . . . . .	53
B.2	No Activity Classes . . . . .	57
B.3	Database Class . . . . .	59
<b>C</b>	<b>Network Plot Method</b>	<b>61</b>
C.1	GraphGenerator.py . . . . .	61
<b>D</b>	<b>Social Unifier Network Graphics</b>	<b>65</b>
D.1	Single User Network . . . . .	66
D.2	Global Network . . . . .	67
	<b>Bibliography</b>	<b>70</b>



## CHAPTER 1

# Introduction

---

The social web is growing at an increasing rate, and it is becoming more and more difficult to manage the large amount of useful available information. The number of commonly used social networks is longer every day, and the evolution of mobile devices keeps making simpler to have access to these social networks and it is making of them the main method of communication. This thesis takes a look at that subject, and along its pages it proposes a solution for the problem of managing different social networks through a mobile device, as it will be described in the following sections.

## 1.1 Problem description

The three most popular social networking sites [1] [2] are Facebook, Twitter and LinkedIn, and that is the reason why they three have been chosen to be the focus of this project. There is a huge amount of valuable information contained in each of those networks, but the large number of connections and vague information on them makes increasingly confusing to manage the resources and take advantage of them. In this project, as a way to enlighten the matter, we will analyze the three social networks with a focus on matching users across them, identifying individuals across platforms.

## 1.2 Thesis aim

This thesis presents two main challenges. First and foremost, the building of a database that will store collected information from social networks, creating an unified graph connected by the junction of connections from every profile. The second challenge is to design an algorithm able to provide potential cross-platform matches from the information collected and stored in the database. The database, therefore, will work as the input for that algorithm, and the detected matches will be stored there, which will modify the graph by unifying the nodes belonging to different networks that contains the profile information of the same single person. In summary, the aim of the thesis is:

- To build a database to store an unified graph connecting the profiles of Facebook, Twitter and LinkedIn users.
- To design an algorithm to match contacts across social networking platforms.

## 1.3 Methodology

For the purpose of the project, a smartphone application will be developed, it will be designed to run on Android 2.2 systems. Through this application, named "Social Unifier", relevant information about contacts of the user will be extracted from each one of the three social networks supported. This means that the Social Unifier system provides to the user a single tool to connect to the APIs of Facebook, Twitter and LinkedIn. This system will include a matching algorithm that will connect with the database to find the matching contacts across platforms, providing to the user a simpler way to understand his networks as a whole and suggesting new connections that may be added to his profile . The database that is meant to store the information is called Borges, and it will be designed over the NoSQL document oriented database technology called CouchDB [3]. Once the system is build, it will be tested to assess the actual possibilities of the aforementioned matching algorithm of getting results by using exclusively the profiles information available through the APIs provided by the social platforms. This tests will also allow to check and study the appearance of the Borges network graph after unifying networks of certain amount of users.

### 1.3.1 Resources

To carry on the practical work of this thesis the following resources have been used:

- One mobile device HTC Desire A8181 with the Android version 2.2 to develop the application.
- An Ubuntu server with Ubuntu 10.04.4 LTS to host the database and other services of the Social Unifier system.

## 1.4 Outline

The thesis is composed of eight chapters and four appendices. The outline of the chapters structure is presented below:

**Chapter 2** describes the first steps of the research, the way to get a first approach to the problem and the ideas and conclusions that were obtained.

**Chapter 3** introduces the database system that has been chosen to achieve the objectives of the project. It presents a brief description of CouchDB systems and details the design of Borges.

**Chapter 4** analyzes the Social Unifier system and its design, including a system overview description and UML case use and sequence diagrams.

**Chapter 5** describes the implementation of the Social Unifier system and the means used to get to its final stage.

**Chapter 6** presents the tests plan conducted to verify the functionality of the product and summarizes the results obtained.

**Chapter 7** discusses the main aspects of this thesis. It analyzes the results of the tests and assesses the development stage of the system, finally, it lays the foundations of the evolution that the project may have in the future.

**Chapter 8** concludes the thesis by assessing the value of this research and the importance of the solutions achieved in regards to the challenges proposed.





## CHAPTER 2

# First Approach

---

At the first stages of the thesis, the idea is to ensure that building an application to fetch the amount of information expected, and to store it on the server, is something viable to be done. It is necessary to find the most appropriate tools and prove their reliability for the project.

## 2.1 Python

As a way to get in touch with the subject and to build a baseline for the development of the project, Python seems to be the optimal programming language to measure the real possibilities of obtaining the desired information from the social networks that will be the objects of study. It is not only chosen for being a great tool to take advantage of the APIs that LinkedIn, Twitter and Facebook provide, but also for its way of handling JSON files and connect with the NoSQL database that will be the core of this thesis.

As it will be detailed in the next sections, a few Python scripts are written to fetch the information from the three social networks and to store it in the database, applying a similar algorithm to the one that will be later developed for the Android application in Java programming language.

## 2.2 Twitter

The first step is the creation of a Twitter application through the developers service of Twitter. Once the application exists, the keys that allow us to send requests to the web API of Twitter are generated. There is a Python package called `Twitter` [4] that provides an extremely simple wrapper around the web API [5], and by using a few simple python scripts, taking advantage of the `Twitter` package for Python and the technology of `Redis` [6], an open source advanced key-value store, it is possible to access to the information of the user of the system, as well as to the information of his followers and the Twitter users who are being followed by him.

The REST API of Twitter has a rate limiting that does not permit more than 350 requests per hour using the same OAuth token. This limitation forces the Python code to be efficient according to the number of requests made. As we are only interested in those connections who may have an actual communication with the user, the script finds the common elements between the followers group and the group of the connections being followed, and it stores them in the database as the actual user contacts, along with all their relevant information, including the connections existing between them, in addition to the relevant information about the Social Unifier system user. The ultimate aim of storing this personal information is to be able to identify those users within the two other social networks.

## 2.3 Facebook

As it has been done with Twitter, it is necessary to register in Facebook as a developer to create the application and get the keys to communicate with the API of Facebook. To facilitate the work of making requests, Facebook maintains an official Python SDK [7] for the Graph API [8], in addition to a query language called Facebook Query Language (FQL) [9], that allows querying Facebook user data with a SQL-style interface.

The python script uses these tools to collect the information about the contacts of the Facebook user who authorizes the use of Social Unifier. Once again, all the accessible relevant personal information about every user and his connections is stored on the database for the purpose of identifying his profiles within the other social networking platforms.

## 2.4 LinkedIn

The same process of getting credentials for the application is followed for linkedIn. As we did before there is also used a Python module [10] to facilitate the interaction with the LinkedIn API [11].

The python script follows the same line followed with Facebook and Twitter, fetching and handling relevant data for the user and his connections and storing it as JSON objects.

## 2.5 CouchBase

CouchDB is a NoSQL open source database that uses JSON to store data as key/value pairs. The next chapter focus on this topic and the reason why CouchDB has been chosen for this project.

Regarding the usability of Couchdb with Python, there is a Python client module called couchdb [12] that allows the user to interact with the database as it is a JSON object. When the scripts that connect with the different social networks get data, they use JSON objects to handle the information and adapt it to the database. The information about the users of the Social Unifier system is stored by using their user names as their entry keys, and the keys used for the rest of the connections are the "ids" they have in the social networks where they belong.

## 2.6 First approach conclusions

The results of the testing the Python scripts and the database in this first approach are quite positive.

Respecting the connection with the APIs of social networks, they were tested by users with more than 500 connections and they proved to be reliable and reasonably fast, taking not more than 10 minutes to handle that amount of users.

The JSON objects, that the database handles, resulted to be the most appropriate way to interact with the information collected from those social networks, greatly reducing the number of needed operations to manage the information

properly and allowing the Python scripts written for the task to be simple and precise.

In short, this first approach confirms that the database system is adequate for the project and that the APIs of the social networks work as it was expected for them to do. In regards to the Python programming language, due to the high compatibility that it has with CouchDB, it was decided that, although the main code and logic of the mobile device application would be written in Java, the matching algorithm would improve its performance if it is written in python.

## CHAPTER 3

# Borges Database

---

One of the main challenges of this thesis is the design of a document-oriented database that simplifies the task of working with JSON objects and presents an easy way to establish connections through Python and Java. CouchBase Single 1.2.0 [13] has been chosen for this purpose, and a database named Borges has been designed to be the core element of the Social Unifier system.

### 3.1 CouchBase

Apache CouchDB [14] belongs to a new generation of database management systems, NoSQL document-oriented databases. CouchBase Single 1.2.0 is the technology of databases used along this thesis, and it is based in Apache CouchDB 1.1.0. The reason for this decision is detailed below, based in a number of factors that makes this system the most appropriate one for a project like this.

#### 3.1.1 Queries, JSON Documents and HTTP requests

CouchDB is formed by a collection of JSON documents stored as key/value pairs, and it uses a map-reduce pattern to index data. In the particular case

of this thesis, where each entry of the database is only identified by the ID of the system user, or the ID of the corresponding social network from where the information for that entry was obtained, this way of querying provides a great usability advantage over the traditional Structured Query Language.

CouchDB replication uses the same REST API that all clients use, and this is why, with the use of JSON documents to handle and structure the information, the process of collecting data from the APIs of the social networks becomes faster and more efficient than using any other system, due to the compatibility of the types of data that they handle.

### 3.1.2 Futon

Futon is the web-based administration console of CouchDB. It can be accessed by a web browser and from there it is possible to create or delete databases and to manage single CouchDB documents. It results to be an intuitive tool to control and understand the database and its structure. Screenshots of it are used along this chapter to show images of how the entries of Borges are structured in Figures 3.1, 3.2, 3.3 and 3.4 .

## 3.2 Design

The concept of the design of Borges lies in the idea of using the database as the input for the matching algorithm and as the input to draw the image of the graph of the whole Social Unifier network. For this, besides containing as much information as possible about the contacts in an easily accessible way, Borges also needs to present a structure that can be readable as a graph.

There are three different types of entries in Borges: Users of the system, social network connections, and Social Unifier information entries.

### 3.2.1 Users

The users of the Android application have to register with an user name and a password to create their profile in the system. Once the registration form is complete, they can collect the information of their contacts from Facebook, Twitter or LinkedIn.

Every entry in the database has a field called "`_rev`", containing the revision information of the document, and a field "`_id`", with the identification key for that entry. In the case of an user of the system the value for that field is equal to the chosen user name. The last essential field in this kind of entries is the one that stores the password of the user. The fields that contain the information collected from every social network are created the first time the user clicks the "get contacts information" button on the application. There are three fields for this matter, "`facebook_contacts`", "`linkedIn_contacts`" and "`twitter_contacts`", that contain the list with the id of every connection the user has in his profile in that network. The last three fields are "`facebook_info`", "`linkedIn_info`" and "`twitter_info`", where every relevant piece of information from the profile of the user in that social network is included. A Futon's view of the entry of an user who has logged in the three social networks is presented in figure 3.1.

Test	ID	Value
	<b>_rev</b>	"kubica"
	<b>facebook_contacts</b>	"299+2254247046901046d8d8d24c05191"
	<b>facebook_info</b>	"{"id":10208, "fbid846, 6109708, 50120106, 50262076, 50260387, 50394266, 50439181, 50439942, 50468160, 50510085, 50545282, 50689267, 50..."
	<b>inspirational_people</b>	"{"id":10208, "fbid846, 6109708, 50120106, 50262076, 50260387, 50394266, 50439181, 50439942, 50468160, 50510085, 50545282, 50689267, 50..."
	<b>birthday</b>	"11/02/1987"
	<b>is_hometown</b>	"{"id":10208, "fbid846, 6109708, 50120106, 50262076, 50260387, 50394266, 50439181, 50439942, 50468160, 50510085, 50545282, 50689267, 50..."
	<b>location</b>	"{"id":10208, "fbid846, 6109708, 50120106, 50262076, 50260387, 50394266, 50439181, 50439942, 50468160, 50510085, 50545282, 50689267, 50..."
	<b>link</b>	"http://www.facebook.com/wojciech"
	<b>education</b>	"{"id":10208, "fbid846, 6109708, 50120106, 50262076, 50260387, 50394266, 50439181, 50439942, 50468160, 50510085, 50545282, 50689267, 50..."
	<b>updated_time</b>	"2012-03-08T17:42:21+0000"
	<b>relationship_status</b>	"single"
	<b>religion</b>	"Regular del Pastafariano"
	<b>id</b>	"102080007"
	<b>first_name</b>	"Jacek"
	<b>timezone</b>	"Europe/Prague"
	<b>username</b>	"wojciech"
	<b>bio</b>	"Nothing is better, no matter where you read it, or who said it, no matter if I have said it, unless it agree with your own reason and your own..."
	<b>email</b>	"wojciech@wp.pl"
	<b>verified_type</b>	"0"
	<b>name</b>	"Jacek Kubica"
	<b>last_name</b>	"Kubica"
	<b>gender</b>	"male"
	<b>favorite_affiliates</b>	"{"id":10208, "fbid846, 6109708, 50120106, 50262076, 50260387, 50394266, 50439181, 50439942, 50468160, 50510085, 50545282, 50689267, 50..."
	<b>linkedin_contacts</b>	"{"id":10208, "fbid846, 6109708, 50120106, 50262076, 50260387, 50394266, 50439181, 50439942, 50468160, 50510085, 50545282, 50689267, 50..."
	<b>linkedin_info</b>	"{"id":10208, "fbid846, 6109708, 50120106, 50262076, 50260387, 50394266, 50439181, 50439942, 50468160, 50510085, 50545282, 50689267, 50..."
	<b>headline</b>	"Senior Technical University student"
	<b>first_name</b>	"Jacek"
	<b>phonenumbers</b>	"{"id":10208, "fbid846, 6109708, 50120106, 50262076, 50260387, 50394266, 50439181, 50439942, 50468160, 50510085, 50545282, 50689267, 50..."
	<b>positions</b>	"{"id":10208, "fbid846, 6109708, 50120106, 50262076, 50260387, 50394266, 50439181, 50439942, 50468160, 50510085, 50545282, 50689267, 50..."
	<b>in_accounts</b>	"{"id":10208, "fbid846, 6109708, 50120106, 50262076, 50260387, 50394266, 50439181, 50439942, 50468160, 50510085, 50545282, 50689267, 50..."
	<b>location</b>	"{"id":10208, "fbid846, 6109708, 50120106, 50262076, 50260387, 50394266, 50439181, 50439942, 50468160, 50510085, 50545282, 50689267, 50..."
	<b>last_name</b>	"de la Peña"
	<b>twitter_username</b>	"{"id":10208, "fbid846, 6109708, 50120106, 50262076, 50260387, 50394266, 50439181, 50439942, 50468160, 50510085, 50545282, 50689267, 50..."
	<b>educations</b>	"{"id":10208, "fbid846, 6109708, 50120106, 50262076, 50260387, 50394266, 50439181, 50439942, 50468160, 50510085, 50545282, 50689267, 50..."
	<b>password</b>	"kubica"
	<b>twitter_contacts</b>	"{"id":10208, "fbid846, 6109708, 50120106, 50262076, 50260387, 50394266, 50439181, 50439942, 50468160, 50510085, 50545282, 50689267, 50..."
	<b>twitter_info</b>	"{"id":10208, "fbid846, 6109708, 50120106, 50262076, 50260387, 50394266, 50439181, 50439942, 50468160, 50510085, 50545282, 50689267, 50..."
	<b>location</b>	"Madrid, Spain"
	<b>status</b>	"{"id":10208, "fbid846, 6109708, 50120106, 50262076, 50260387, 50394266, 50439181, 50439942, 50468160, 50510085, 50545282, 50689267, 50..."
	<b>profile_image</b>	"https://ad.twimg.com/profile_images/1249159781/wojciech_445x400px.jpg"
	<b>created_at</b>	"2012-03-08 16:51:07+0000"
	<b>statuses_count</b>	44
	<b>lang</b>	"en"
	<b>id</b>	1518403
	<b>background_image</b>	"https://ad.twimg.com/images/cover/fig-61"
	<b>time_zone</b>	"Europe/Prague"
	<b>favourites_count</b>	0
	<b>description</b>	"{"id":10208, "fbid846, 6109708, 50120106, 50262076, 50260387, 50394266, 50439181, 50439942, 50468160, 50510085, 50545282, 50689267, 50..."
	<b>friends_count</b>	45
	<b>name</b>	"Jacek"
	<b>screen_name</b>	"Jacek Kubica"
	<b>followers_count</b>	49
	<b>listed_count</b>	0

**Figure 3.1:** View of the entry of a Social Unifier user in Futon

### 3.2.2 Connections

The Borges entries for the connections follow a similar structure to the one followed for the user entries. The "`_id`" field, in this case, takes the value from the id of the profile of that connection in the social network where it belongs. Besides that, there is a `contacts` field, which contains a list of its connections among the other entries in the database that come from the same social network

and have at least one of the users of the system as a common friend. The last field is the one containing the data about the profile that were available through the API of the corresponding social network. The amount of information stored depends on the privacy controls of that network and the actual information that the owner of the profile is willing to share. A view of the connection entries in Futon for each one of the three social networks is displayed in Figure 3.2, 3.3 and 3.4.

Field	Value
<code>_id</code>	"100000198537629"
<code>_rev</code>	"3-4ebcb9b5ac8447095a094bf08e8ddc0"
<code>facebook_contacts</code>	"[6109708, 503260367, 505109885, 506695287, 568246302, 630826755, 695375569, 707310153, 709867376, 712420331, 803995549, 100000897109327, 100..."
<code>facebook_info</code>	<pre> id "100000198537629" birthday "07/18" first_name "Isabel" username "isabel.leal.98" hometown   id "106504859386230"   name "Madrid, Spain" location   id "106504859386230"   name "Madrid, Spain" locale "es_LA" link "http://www.facebook.com/isabel.leal.98" name "Isabel Leal" last_name "Leal" gender "female" education   updated_time "2012-05-12T09:20:38+0000" </pre>

Figure 3.2: View of the entry of a Facebook contact in Futon

Field	Value
<code>_id</code>	"391560183"
<code>_rev</code>	"2-ac70416ce043002285fd0fc0236a7502"
<code>twitter_contacts</code>	"[50637206, 566470895, 507846926, 375038640, 55394073, 418631917, 485914703, 18135626, 320711793, 491325176, 75318282, 431786421, 43420608, 2..."
<code>twitter_info</code>	<pre> status "StatusJSONImpl(createdAt=Wed May 09 11:43:03 GMT+02:00 2012, id=200158934064037888, text="Buenos días http://t.co/f6ULeKxp", source="&lt;a href..." location "madrid" profile_image "http://a0.twimg.com/profile_images/1970241429/3XXc114v_normal" created_at "Sat Oct 15 21:01:24 GMT+02:00 2011" statuses_count 149 lang "es" background_image "http://a0.twimg.com/images/themes/theme6/bg.gif" id 391560183 favourites_count 0 friends_count 111 description "" name "Naiara Campo" screen_name "Nai_ai_ai" followers_count 46 listed_count 2 </pre>

Figure 3.3: View of the entry of a Twitter contact in Futon



Field	Value
<code>_id</code>	<code>"-h4MtERBzX"</code>
<code>_rev</code>	<code>"42-1a7012b3fc6d8f8e6e4bc1d16c2e527e"</code>
<code>linkedin_contacts</code>	<code>"[]"</code>
<code>linkedin_info</code>	<code>{</code> <code>  id: "-h4MtERBzX"</code> <code>  headline: "Estudiante en Universidad Europea de Madrid"</code> <code>  first_name: "Sergio"</code> <code>  phonenumbers: []</code> <code>  positions: []</code> <code>  im_accounts: []</code> <code>  location:</code> <code>    last_name: "Ramírez"</code> <code>    twitter_accounts: []</code> <code>    educations: ["Universidad Autónoma de Madrid"]</code> <code>}</code>

**Figure 3.4:** View of the entry of a LinkedIn contact in Futon

### 3.2.3 Social Unifier Information

There exist four entries in Borges whose function is to facilitate the handling of the structure of the database. Three of them, one for every social network, are meant to keep a record of the users of the system who have collected their contacts from that network. The keys of those entries are "tw\_users", "fb\_users" and "ld\_users". Every field they contain uses the id of an user of that social network as the key, and its chosen user name for the Social Unifier system as the value, this way, it is possible to find an user in the system by not only looking for his user name, but also by looking for the id of his profile in the different social networks. The fourth one of the social unifier information entries is named "matches", it stores contacts matched across networks with the matcher algorithm. The key of each field is the id of a connection from one of the social networks, while the value stored is a list, with a maximum of two elements, that is composed by the ids of the profiles that that person has in the other two networks supported by the system. These entries allow the database to be managed, reducing as much as possible the number of queries made and the number of entries that need to be modified. The matches record gives the possibility of finding the profiles belonging to a contact within other networks, without having to keep that information stored inside of the entries that identify those profiles.

### 3.2.4 Graph

As it is expected to be possible to draw a graph that represents an image of Borges, the database has been designed in a way that can be read as a collection of nodes and edges. Any algorithm whose intention is to obtain the graph contained in Borges should consider each user of the system, each person included

in the matches field and each connection entry that is not in the matches record as a node. The edges between those nodes will be found in the connections field of every entry, each id on those connections lists represents an edge between that entry node and the node of the entry with the id found on the list. In the case of adding edges for Social Unifier users or for found matches, the social unifier information entries will work as a dictionary to find the key of the node comprising that connection id. If it is looking for an user, it will find his user name in the corresponding entry with the list of that social network users. If it is looking for a match, it will find the list with the other keys representing that same person in the matches entry. Appendix C contains the description of a way to read and plot the graph of the system and in appendix D it is possible to see some figures illustrating the Borges database graph.

## CHAPTER 4

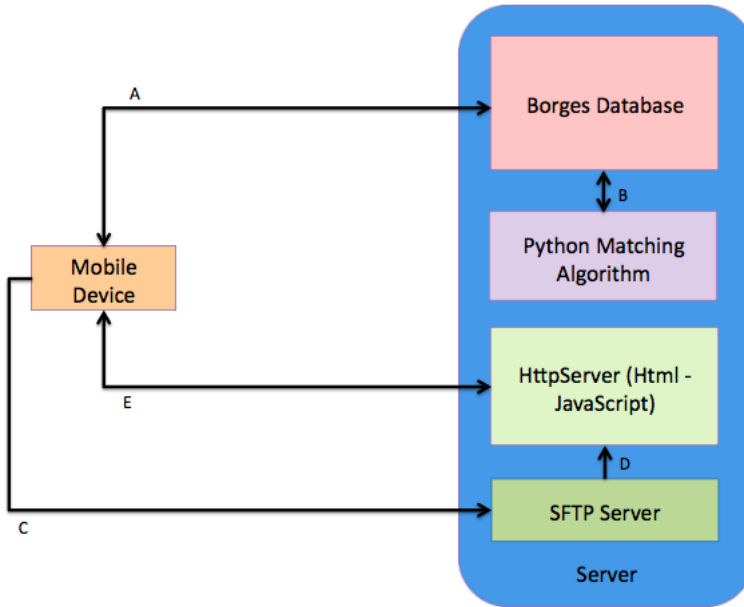
# Analysis and Design

---

The Social Unifier system is the answer given to the problem proposed in this thesis. This chapter focuses on the structure of this system and it provides an analysis of its different components and the way they interact with each other. It begins with an overview of the whole system, then a deeper description of the server and the Android application and it finishes with a summary of the ideas presented in the chapter.

### 4.1 System Overview

The Social Unifier system is composed by a mobile device application that connects with the services provided by a server. Those services are the Borges database, the python script that runs the matching algorithm, a SFT server and a HTTP server. Figure 4.1 illustrates the architecture of this system.



**Figure 4.1:** System overview of Social Unifier (The components and connections between them are described in sections 4.1.1 and 4.1.2)

### 4.1.1 Components

**Mobile Device:** An Android application has been developed for the user to interact with the system and manage all the services included on it, which are allocated on the server. The application is designed for the Android 2.2 platform, and its design will be the subject of the next section of this chapter.

**Borges Database:** The Borges database, whose design was the matter of the previous chapter, is allocated on the server and provides its own HTTP server to handle requests.

**Python Matching Algorithm:** There is a Python script running in the server to establish a connection with the Borges database and traverse through it, checking every entry, to find single persons with profiles in different social networks and to store those findings into the "matches" field of the database.

**SFTP Server:** The SFTP server allows the mobile application to upload files to the server of the system. Those files contain the graphs of the networks that

the user wants to be able to browse, and they will be read by the JavaScript script that will be accessible through the HTTP server.

**HTTP Server:** This HTTP server is meant to receive HTTP requests from the browser of the mobile device. It shows the html page containing the JavaScript code that prints the graph plot of the network requested by the user. That script uses the JavaScript library "d3.js" [15] to manipulate the graph files and draw the graph.

#### 4.1.2 Communication between Components

**A** The mobile device sends HTTP requests to the database and receives the responses as JSON documents. There are four different kinds of requests: Read, Write, Update and Delete, which are made by using GET, POST and DELETE commands.

**B** The python scripts connects with the database through the CouchDB library for python to which we referred in chapter 2. This library manages the requests and allows python to interact with the database as it is a dictionary object, simplifying the task of reading and writing on Borges.

**C** The mobile device uploads the graph files to a folder inside of the server by setting a SFTP connection. A Java library called Java Secure Channel (JSch) [16] is used for this task.

**D and E** When the browser of the mobile device sends requests to the HTTP server, the server sends, as a response, the file "index.html". That html file includes the JavaScript code that draws the graph of the network, by using the "d3.js" JavaScript library. The methods from the library will read the graph files that the user chose to be drawn, which was uploaded through the SFTP server.

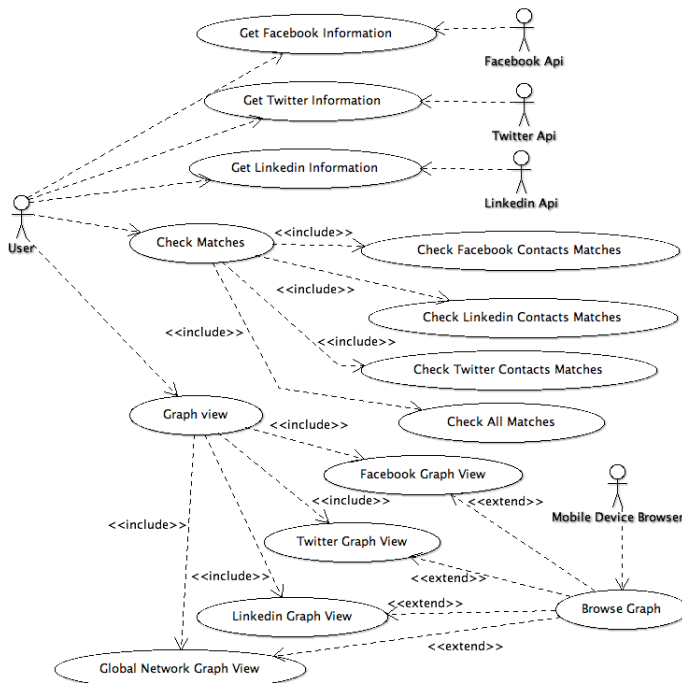
## 4.2 Mobile Device Application

The Android application designed must be able to provide a tool that allows the user to collect information from the social networks that he is signed in, it also gives the user the possibility of checking the graphs of those networks, the graph of the combination of them, and the matching profiles that were found across the social platforms, together with the suggested connections that may

also be found. To introduce that design, an UML use case diagram and an UML sequence diagram are shown and explained below. The section finishes with a brief discussion about the aesthetic design of the user interface of the application.

### 4.2.1 UML Use Case Diagram

Figure 4.2 shows an UML use case diagram that presents the basic interactions of the Social Unifier system with the user and the other external actors involved, namely, the social network APIs and the browser of the mobile device.



**Figure 4.2:** Social unifier UML case use diagram.

The diagram shows the options that the user has to interact with the system after he has registered and logged in. We can classify these options in three groups:

**Collecting data:** The user can choose the option of getting the information about his profile in LinkedIn, Twitter and Facebook, and add it to the database together with the information of his contacts. To do this, the system must connect to the API of the selected social network.

**Check matches:** This choice gives to the user the possibility of checking the contacts matched across his social networks, and, in the case of the option "Global matches / Suggested connections" that is included in the main menu, it is also possible to check the profiles that its contacts have in other social platforms but are still not connected to the user. The matching algorithm is running in the server and it is not synchronized with the mobile device, so it may take a while for the matches to be found once the user has collected the contacts information. The application is able to display only the matches for the contacts in one of the three single social networks or the total of matches found among the three of them, according to the choice of the user. In the last case, it will always also show the list of suggested connections.

**Graph viewing:** It is possible to check a graphical view of the graph composed by the connections of the user. The application will open the default browser of the mobile device and it will display the web page including the drawing of the graph selected by the user. That user can choose between visualizing the global graph for his whole network or one composed only for the contacts belonging to one of the three social networks.

### 4.2.2 UML Sequence Diagram

Figure 4.3 illustrates an UML sequence diagram representing how processes interact with each other and the order of those interactions for the use cases shown above. It works as follows:

The first time the user starts the application, he will have to register. The application will check the Borges database to ensure that the user name chosen is available, and after getting a response, the new user entry with the information about the registration will be stored in Borges. When the registration is complete the user is allowed to log in by using his user name and password, which will be checked with the ones stored in the database to grant permission to use the system.

When the user chooses to get the information from one of the social networks, the first step is to login on that social network and to allow Social Unifier to have access to the information of his profile. This OAuth dance [17] with the API of the social network provides the application with an access token. The access token is used to make requests asking for the information about the user and its connections. Once Social Unifier receives the response with the data, it connects with the database to request for the former information about that user and that social network, which might be already stored. Finally, the algorithm updates the JSON documents gotten from Borges with the new data provided by the social network.

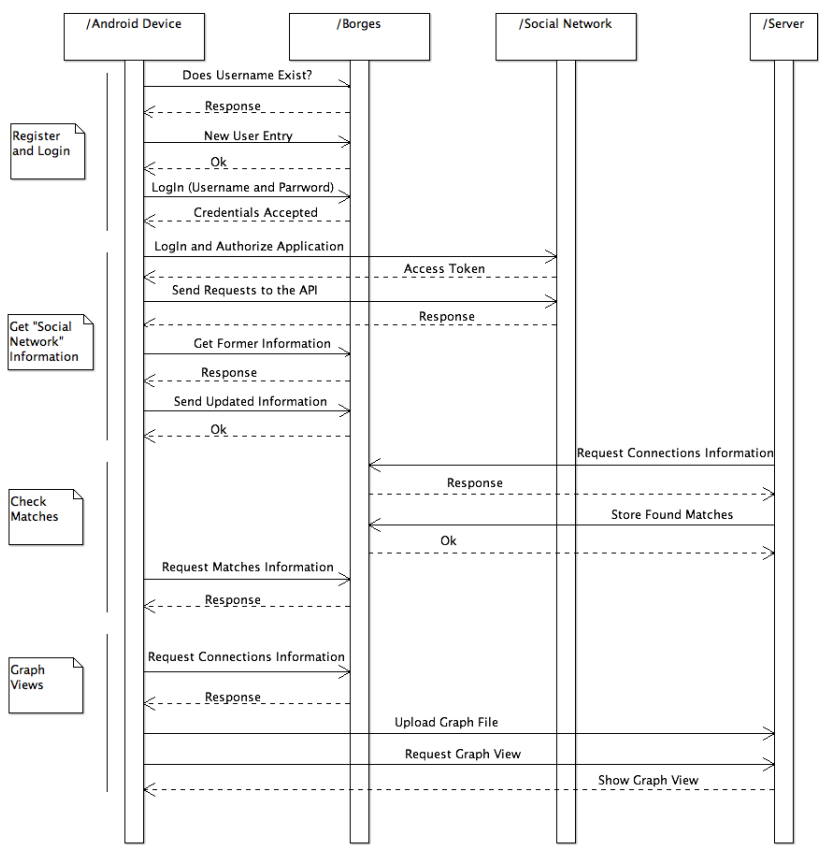


Figure 4.3: Social unifier UML sequence diagram.

The server keeps a python script running the matching algorithm through the database. This script reads entries from the database, it finds new contacts



whose profiles match across networks and store those matches back into Borges. When the user chooses to check the matches information, the mobile device requests the document of the entry containing the matches to Borges and it reads the response to display a list containing the matches found for the contacts of that user.

The last operation shown in the diagram is the one that shows the graph plot of the network of an user. The application reads the database to get the contacts and it composes the graph, once the graph is generated, it creates a file containing it and uploads it to the server through the SFTP server. Finally, the browser of the application makes the HTTP request to the server to display the image of the graph.

### 4.2.3 User interface

Appendix A contains the mobile device screenshots, where it is possible to see the graphical design of the Android application. This design takes advantage of the items included in the palette of the basic android graphical layout.

## 4.3 Summary

What was described in this chapter can be summarized in a few lines. The system comprises three main components, namely, the Android application, the Borges database, and the server, which includes a HTTP server and a SFTP server. Those components interact with each other mostly through HTTP requests, and they connect the same way with the APIs of Facebook, LinkedIn and Twitter, by carrying out the Oauth dance to get an access token. Next chapter focuses on the task of implementing this components.



## CHAPTER 5

# Implementation

---

The process of implementing the design explained above is detailed in this chapter. The prototype described below is the one that has been used to conduct the tests and the one that provides the results that will be discussed in chapters 6 and 7. The following sections take a special look to the problems that have arisen while implementing the Social Unifier system and the way to solve them, including the overcome challenges and non trivial code snippets.

## 5.1 Overview

The process of implementing the system can be fragmented into the following sections, which form the outline of this chapter:

- Interface
- Social Networks
- Connection to the Server
- Check Matches / Suggested Connections

- Graph View
- Matching Script

The five first parts of this outline describe elements that have been implemented in Java programming language, using Eclipse Indigo Service Release 2, Eclipse IDE for Java Developers [18] and the Android SDK [19] for the Android platform 2.2 [20] as the software needed to develop and build the program. Appendix B contains figures that illustrate the classes overview of this Java code. On the other hand, the matching script has been written in Python Programming Language using for that the Python 2.7.2 Release [21]. Finally, besides Java, the code that draws the graph plot, which is a part of the graph view system, is written in JavaScript scripting language.

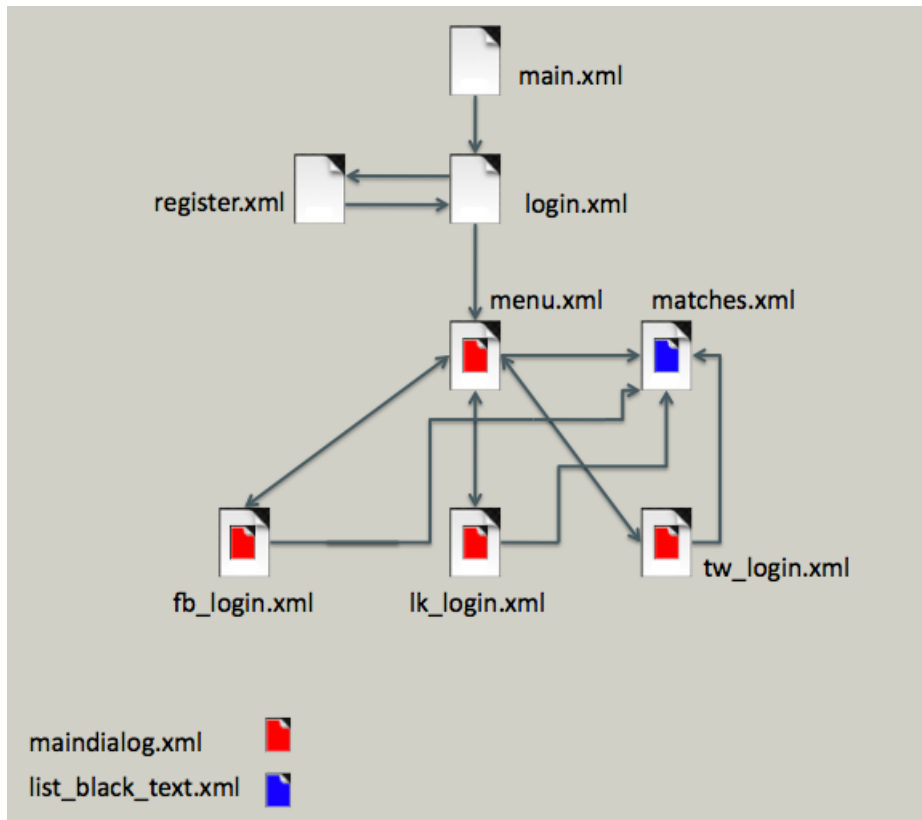
### 5.1.1 Eclipse Project

As said before, Eclipse includes the Android SDK for the Android platform 2.2. The project target is set to be Android 8. This project has twelve activities declared in the "AndroidManifest" file, and all of them are declared to be portrait oriented, in order to prove an uniform design to the application. The project contains two source packages, the first one, "thesis.unifier", contains the bulk of the classes of the application, and the other one, "org.base", contains the class providing the methods to connect to the Borges database. In addition to the Android SDK, the project includes the Facebook Android SDK library [22], and integrates several other Java libraries that will be mentioned in the following sections.

## 5.2 Interface

The project contains ten layouts that comprise the user interface of the application. The screenshots illustrating the result of those layouts are included in Appendix A. Those layouts are ".xml" files declaring the elements that will compose the different views of the application. Together with these ten layout files there is another XML file called "strings.xml" that contains the text strings that will be used by the elements declared in the layouts. Figure 5.1 illustrates the usage cycle of those layouts during the execution of the application. Arrows in the picture indicate the possibility of going from one layout to another one, by using elements included in the first layout such as buttons or image buttons. The files "maindialog.xml" and "list\_black\_text.xml" are the only ones

not associated to any activity class. The first one generates an Android custom dialog and it is used over the layouts generated by "menu.xml", "fb\_login.xml", "lk\_login" and "tw\_login.xml". The second one is the layout for items inside of a list and it is used over the "matches.xml" layout.



**Figure 5.1:** Layouts usage throughout the Social Unifier life cycle.

## 5.3 Social Networks

The three activities that carry on the connection with the three social networks APIs use a similar algorithm. The aim of these classes is to provide a Single Sign-On access control for the Social Unifier system.

### 5.3.1 Facebook

The tools that Social Unifier needs to connect to the Facebook API are the Facebook Android SDK and the activity class "Fb\_main". This class takes advantage of the resources provided by the SDK to set a connection and make requests. The access token given by the social networking site is stored using the "shared preferences" class, and it allows the user to use Social Unifier without having to authorize the access to the Facebook information by the application every time a new connection is set. It has been only necessary to implement two methods inside of the "Fb\_main" class, these are the method "onCreate" and the method "onActivityResult". The first of them contains the bulk of the code, including the algorithm to collect the contacts information, while the second one handles the result of returning to this activity after executing a different one.

### 5.3.2 Twitter

There have been implemented five classes that provide the methods to connect to the Twitter API. Those classes are the activities "Tw\_main" and "PrepareRequestTokenActivity" and the classes "TwitterUtils", "OAuthRequestTokenTask" and "RetrieveAccessTokenTask". The last two are an extension of "AsyncTask<Void, Void, Void>" and the latter is contained inside of the "PrepareRequestTokenActivity" class. They use the library OAuth-Signpost [23] to perform the OAuth dance and establish a Single Sign-On connection with the Twitter API, and "Tw\_main" and "TwitterUtils" also use the library "Twitter4j" [24] as a wrapper around the API to make relevant Twitter requests. Just as it was done with Facebook, the access token is stored in the mobile device to make a just one time task for the user to go through the authorization process.

The class "Tw\_main" generates the user interface for the menu and "PrepareRequestTokenActivity" generates the authentication interface. On the other hand, "TwitterUtils" contains a set of methods that will be called from the "Tw\_main" class, those methods are "isAuthenticated", to check the existence of a valid access token and "getInfo", "getUsersInfo" and "getFriends", to collect the relevant information about the network connections. Finally, "OAuthRequestTokenTask" and "RetrieveAccessTokenTask" are called from "PrepareRequestTokenActivity" and they contain the methods responsible for conducting the OAuth authorization flow.

### 5.3.3 LinkedIn

The activity class "Lk\_main" generates the user interface to connect with the LinkedIn API, to do this, it uses the "Linkedin-j" Java library [25], which, in turn, takes advantage of the OAuth-Signpost library to perform the OAuth dance. It proceeds as it was described in the sections above, setting a Single Sign-On access control and saving the access token information by using the Shared Preferences class.

The "Lk\_main" class provides a method called "getData" that performs the task of making requests to the API to collect the connections information. The methods "startAuthenticate" and "finishAuthenticate", on the other hand, carry out authentication tasks. To begin, "startAuthenticate" is called inside of the "onCreate" method and "finishAuthenticate" is used from the "onNewIntent" method, to give closure to the authentication process.

## 5.4 Connection to the Server

Social Unifier connects directly to the Server through two different methods, it contacts the Borges database through HTTP requests and it needs to upload files to the SFTP server. The two sections below details the classes implemented to perform these tasks.

### 5.4.1 Database Class

The class "Database", included in the package "org.base", works as a wrapper around CouchBase providing the functions necessary to manage the data stored. This class is used from every other class in the Social Unifier application that has to interact with Borges. The class has two constructor methods, one in case the database needs to authenticate the credentials to be accessible and another one in case it does not. Social Unifier creates instances of the class "Database" by using the constructor that requires credentials, because of the authentication particularities of Borges. The methods implemented on this class are the following:

The **read** method performs HTTP "Get" requests to read from the database the value associated to a given key.

The **write** method performs HTTP "Post" requests to sent a value to be stored

into the database, that value will be associated to a key given as a parameter of the write function.

The **upload** method uses the method read to get a document from the database, and, after updating the information contained into that document, it stores it again with the same key by using the write method.

The **delete** method performs HTTP "Delete" requests to delete the entry of the database that matches the key given as a parameter of the method.

The set consisting on the four previous methods provides all of the tools that Social Unifier needs for its interactions with the database and it has proved to be a highly portable tool. Its has been designed with the intention of being a highly scalable class that could be the starting point for a more complete CouchDB Java library.

### 5.4.2 Upload Class

The class "Upload" performs the task of uploading graph files to the SFTP server of the system. This class uses the above mentioned library, Java Secure Channel (JSch), to stablish the connection given the host name, the port, and the user credentials. The class contains a single method, "up", which creates a session and opens a channel to upload the file into a given path inside of the server. The file is converted into a "ByteArrayInputStream", and that will be the object sent through the open channel.

## 5.5 Check Matches

There are four activity classes intended to calculate and show the matching connections found by the algorithm, "G\_Matches", "FB\_Matches", "TW\_Matches" and "LK\_Matches". The first one displays the matching profiles found for every connection of the user in every social network, and in case not all of the profiles matching that connection are connected to the application user, it will present them as "suggested connections". The remaining classes display the matches found for every connection of the users profile in a particular social network. The algorithm of these classes is included inside of the "onCreate" method, where they present a screen containing a "ListView" that shows the contacts matched and a message reporting the ratio between matches found and the total amount of possible hypothetical matches. This ratio is calculated by assuming that the



contacts of the social network with the smallest number of user connections will also have profiles in the other two social networks, and that the contacts of the social network with the second smallest number of user connections will also have profiles in the social network with the longest number of user connections. "G\_Matches" will also include a message reporting the number of suggested connections found.

Following the assumptions above, to calculate the success ratio for the matching profiles found for every connection in every social network, the hypothetical maximum of possible matches would be the result of the following algorithm:

```

if  $FbContacts < LdContacts \wedge FbContacts < TwContacts$  then
  if  $LdContacts < TwContacts$  then
     $maximum = FbContacts * 2 + LdContacts$ 
  else
     $maximum = FbContacts * 2 + TwContacts$ 
  end if
else if  $LdContacts < FbContacts \wedge LdContacts < TwContacts$  then
  if  $FbContacts < TwContacts$  then
     $maximum = LdContacts * 2 + FbContacts$ 
  else
     $maximum = LdContacts * 2 + TwContacts$ 
  end if
else
  if  $LdContacts < FbContacts$  then
     $maximum = TwContacts * 2 + LdContacts$ 
  else
     $maximum = TwContacts * 2 + FbContacts$ 
  end if
end if

```

When calculating the success ratio for the matching profiles found for only the connections from a specific social network (Facebook, in this particular example), the hypothetical maximum of possible matches would be the result of the following algorithm:

```

if  $FbContacts < LdContacts \wedge FbContacts < TwContacts$  then
   $maximum = FbContacts * 2$ 
else if  $FbContacts < LdContacts \wedge FbContacts > TwContacts$  then
   $maximum = FbContacts + TwContacts$ 
else if  $FbContacts > LdContacts \wedge FbContacts < TwContacts$  then
   $maximum = FbContacts + LdContacts$ 
else
   $maximum = TwContacts + LdContacts$ 
end if

```

## 5.6 Graph View

To implement the graph view system for Social Unifier, there has been written a class called "Graph" that includes a single method, "generateGraph", which goes through the Borges database and returns a JSON Object containing the information about all the nodes and edges in the graph to be shown. The algorithm implemented is intended to make a difference among contacts from different social sites when drawing the graph, to do this, "generateGraph" checks the "matches" entry in the database and assigns a different "group" value to every node. There are seven possible groups: Facebook, Twitter, LinkedIn, Facebook + LinkedIn, Facebook + Twitter, LinkedIn + Twitter and Facebook + LinkedIn + Twitter. The method receives a parameter to select if the graph to be composed must include every contact of the user or just the ones belonging to a particular network. The resulting JSON object is what will be uploaded as a file into the server to be read by the JavaScript code that will draw the graph image.

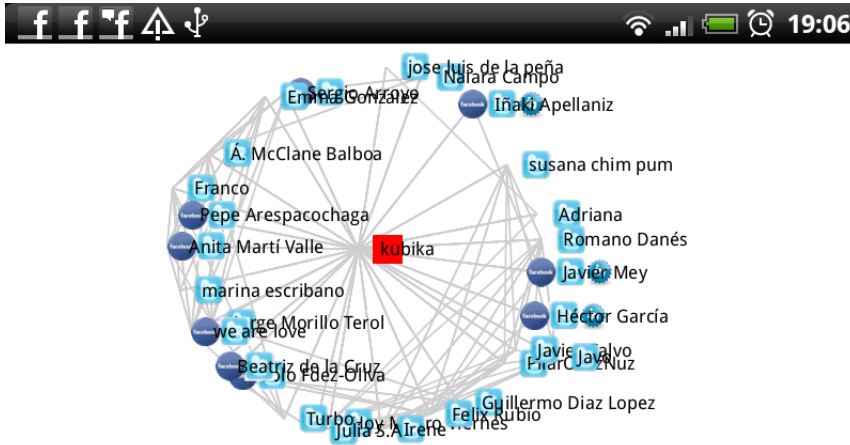
### 5.6.1 JavaScript

The file "index.html" located in the server contains a JavaScript code that uses the "D3.js" library to draw the graph uploaded by the Social Unifier mobile application. When the browser opens the web page, the script reads the graph file and builds the image by adding every node and edge given by the JSON document contained in the graph file. In that JSON document, the edges have a "source" and a "target" field, to indicate the nodes connected by them, and the nodes themselves have a "group" field, indicating the group they belong among the seven ones discussed above. According to the group value, the algorithm associates a different icon to each node to be drawn, so it is possible to identify in the image in what social network or networks the person represented by that node has a profile.

### 5.6.2 Graph View Discussion

Depending on the mobile device characteristics and the browser used, the graph view system may not work properly. Mozilla Firefox for Android is recommended to be set as the default browser of the device for a correct system operation, due to its capability of rendering the SVG images that will be produced by the "D3.js" JavaScript library. As for the performance of the phone, when the amount of nodes in the graph is sufficiently high, it is possible that the

browser has problems rendering the image and it takes more than the expected amount of time to do it. Figure 5.2 shows a screenshot from an Android device that is using the Mozilla Firefox browser to display the graph representation of Twitter connections of a Social Unifier user.



**Figure 5.2:** Screenshot of the Graph view of Twitter connections of an user.

## 5.7 Matching Script

The file "matcher.py" contains the Python script that finds matching profiles across different social networks. This script runs in the server and accesses Borges through the Python couchdb module. The matching algorithm can be described as a loop that reads the connections list of every application user and compares the profile information of every contact from one of the social networks to the profile information of every contact in the two other networks, and in case it finds a match, it adds it to the "matches" JSON Object, besides that, it also stores the profiles of all Social Unifier users as matching profiles. After that, it replaces the former "matches" entry in Borges with the updated new one, and this process is repeated every half hour.

Due to the privacy restrictions of the social networks APIs, the relevant information that can be used to identify a match among two profiles by the comparison of its fields is very limited, and it will be a topic of discussion in chapter 7. That

information is reduced to the following:

**Facebook and Twitter:** Profiles could be matched according to the full name of the user, his user name (Facebook) and screen name (Twitter), his time zone and his location. But despite this, the only fields that are actually used for the comparison are the full names and the user name and the screen name.

**Facebook and LinkedIn:** Profiles could be matched according to the full name of the user and his location, but, for the same reasons than above, the only fields that take part in the algorithm are the full names of users.

**LinkedIn and Twitter:** As it happened with Facebook and LinkedIn, profiles could be matched according to the full name of the user and according to his location, but the full name of users is the only element being compared to find these matching profiles.

# Test and Results

---

A test plan has been designed and conducted to check the Social Unifier system functionalities and its reliability. This chapter presents a brief description of this test plan and a raw display of data obtained as results from the tests. Those results will be discussed and analyzed in chapter 7.

## 6.1 Test Plan

To test the Social Unifier system, it has been used by ten international DTU students under different circumstances, to wit, different Android mobile devices and different internet connection characteristics. All of the test subjects have Facebook profiles, seven of them have Twitter profiles and seven of them have LinkedIn profiles. The aim of these tests, besides checking the robustness of the system, is to assess the accuracy of the matching algorithm and the suggested connections algorithm as well as the response of the Borges database to the storage and organization of certain amount of information. Finally, it is interesting to see the state of the global network created on the system after fetching the contacts information of every test subject.

## 6.2 Matching Results

The information about users connections that has been obtained through the test plan is shown in tables 6.1, 6.2, 6.3 and 6.4. These tables contain the number of connections that each user has in each social network, besides the number of matching profiles found for those connections. Table 6.4 also contains the information about the number of suggested connections for every user. In every table the matches are expressed as the ratio between the number of matching profiles found and the number of possible matches for that user. The number of possible matches is calculated following the algorithm explained in chapter 5, and although it is not equal to the actual value of possible matches, which is expected to be significantly lower but can not be figured, this number intends to give a rough idea about the accuracy of the matching algorithm. According to data included in the four tables we can state the following information:

- The matching algorithm presents a success rate of at least 30% for Facebook connections, 25.89% for Twitter connections and 20.68% for LinkedIn connections. The success rate over the whole Social Unifier network is at worst 25.16%.
- Users receive an average of 8.2 suggestions for new connections for each 380.5 contacts that they already have.
- The Social Unifier network includes 3805 connections after being used by ten users, of those connections 90.46% are Facebook profiles, 5.34% belong to Twitter and 4.20% belong to LinkedIn.
- Although there are 3805 users connections in the system, the number of profiles stored in the Borges database is lower than that, as it is explained in the next section. This is due to the existing shared connections between the students that took part on the tests.

## 6.3 Network

Once the tests have been conducted, Borges database contains 44.7 megabytes of information, resulting in 2710 rows. Four of those rows are the Social Unifier information entries, and ten of them are the application users entries, therefore, this means that the amount of contact profiles collected and stored in the database is 2696, compared to 3805 existing connections. What this information suggests is that 1109 of the application users connections are mutual contacts with some of the other nine test subjects. In the same way, the total amount

of pairs of matching profiles found and stored in Borges is 94, but the total of matches shown in the Android application to the test subjects is 118, which means that some of those 94 matches are connected to more than one of the application users.

After finding the matching connections of the system, the graph formed by the profiles contained in Borges has 2667 nodes and 23732 edges. Each node of the graph represents a profile from one of the three supported social networks, or, in case of the user nodes or the matching profiles nodes, it represents a single person who may have profiles in more than one of the social networks. Likewise, the edges of the graph are the graphic representation of each existing connection between two profiles stored in Borges. Appendix C describes the method used to obtain this graphic representation of Borges, and Appendix D contains Figure D.2, displaying the graph plot of the network. Some interesting facts can be obtained by analyzing the graph:

- The graph contains five different node types, namely, users, Facebook profiles, LinkedIn profiles, Twitter profiles and matching profiles. As it is expected to be, there are not existing edges between Twitter and LinkedIn nodes or between LinkedIn and Facebook nodes, but it is possible to appreciate that there are two edges from Twitter nodes to Facebook nodes, this issue will be discussed in the next chapter.
- There are 2696 entries in Borges but just 2667 nodes in the graph, this is because of the several profiles that match, whose matches have been reduced to a single node in the graph representation.
- The average degree of the graph is 17.797. That number represents the average number of connections that every user, matching profiles or profile in the system has.
- The network diameter is 6, so the further a person with a profile on the system will be from meeting any other person on the system is a distance of 5 persons.

User	Facebook contacts	Matches / Possible matches
User 1	574	24 / 54
User 2	320	21 / 59
User 3	338	14 / 40
User 4	557	32 / 134
User 5	331	5 / 9
User 6	308	4 / 9
User 7	252	2 / 31
User 8	168	2 / 13
User 9	254	5 / 13
User 10	340	0 / 1
Average	344.2	10.9 / 36.3

**Table 6.1:** Facebook tests data

User	Followers	Friends	Twitter contacts	Matches / Possible matches
User 1	49	45	28	17 / 54
User 2	67	126	59	21 / 59
User 3	29	62	19	12 / 38
User 4	99	239	74	21 / 134
User 5	-	-	-	- / -
User 6	16	29	9	4 / 9
User 7	-	-	-	- / -
User 8	3	16	1	0 / 2
User 9	14	29	13	5 / 13
User 10	-	-	-	- / -
Average	39.57	78	29	11.43 / 44.14

**Table 6.2:** Twitter tests data



User	LinkedIn contacts	Matches / Possible matches
User 1	26	17 / 52
User 2	-	- / -
User 3	21	10 / 40
User 4	60	19 / 120
User 5	9	5 / 9
User 6	-	- / -
User 7	31	2 / 31
User 8	12	2 / 13
User 9	-	- / -
User 10	1	0 / 1
Average	22.86	7.86 / 38

**Table 6.3:** LinkedIn tests data

User	Total connections	Matches / Possible matches	Suggested connections
User 1	628	29 / 80	5
User 2	379	18 / 59	9
User 3	378	18 / 59	3
User 4	691	35 / 194	3
User 5	340	5 / 9	13
User 6	317	2 / 9	9
User 7	283	4 / 31	13
User 8	181	2 / 14	4
User 9	267	5 / 13	12
User 10	341	0 / 1	11
Average	380.5	11.8 / 46.9	8.2

**Table 6.4:** Global Social Unifier network tests data



## CHAPTER 7

# Discussion

---

The interpretation of the results above, the analysis of the current state of the network and the discussion of the stage of development of the system are necessary at this point to form an idea of the achievements of this master thesis before closing this report with the conclusions that have been obtained from it. The next sections summarize the benefits that the Social Unifier system generates and its capability of providing answers to the proposed problem, as well as the possible future evolution lines of the project.

### 7.1 Results Analysis

The first issue that draws attention from the tests conducted is the differences between the amount of Facebook connections and the amount of Twitter and LinkedIn connections. Of all users of the application only 70% of them use LinkedIn and only 70% of them use Twitter, however 100% of them have Facebook profiles. The conclusion obtained from these data is that international DTU students are in general not yet fully involved in the labor market nor have a big influence or interest in Twitter.

Regarding the matching profiles, the average of matched profiles is 25.16%, therefore, for each four profiles that might have a matching profile in some of the

other two social networks, the system finds one. The idea has been to find these matches by taking only advantage of profiles information that is available through the social networks APIs, but the only unique identifiers that can be obtained are the Facebook user name and the Twitter screen name, which is not enough information, so this makes necessary the use of other fields from the profiles to find the matches. It has been decided that the matching algorithm determines if the profiles belong to the same person by comparing the profile full name of them, besides the user name and the screen name mentioned above. There were some other profile fields that could have been used to ensure that the profiles actually belong to the same person, instead of belonging to two persons with the same name, such as the location or the education, but the success rate significantly decreases by adding these fields to the algorithm, due to the vagueness of the information they contain. With the used method, although the amount of matches found is higher, it is also possible that the system matches two profiles which do not actually belong to the same person. To assess the probability of this happening, the 94 matches stored in Borges have been carefully checked, obtaining as results that all of the matching profiles found are correct. This means that, according to the tests conducted, the possibility of finding wrong matching profiles is not higher than 1.06% at worst.

One of the main goals of the project was to evaluate the capability of the CouchDB technology to handle this kind and amount of information and to establish connections with each part of the system, specially the Android mobile device. Either the CouchDB Python module or the CouchDB Java Library developed for the project have proved to provide a highly reliable connection to Borges, what has been translated into a flawless performance when storing and managing information. As planned, it has been possible to create and read the graph of the network from Borges, containing all the profiles information. As the graph, according to the results described in the previous chapter and to Figures D.1 and D.2, it is fair to say that it meets expectations, and it is predicted that as the network keeps growing and evolving, the appearance of the graph plot will remain very similar to the present one.

## 7.2 Development Stage of the System

The fact of conducting the test plan demonstrated that the operation of the Social Unifier system is not perfect and there are a few aspects that should be improved in order to achieve a better performance in the future. Fortunately these issues do not have an actual impact over the test results but it is important to identify them and describe the way to solve them.

- As pointed out in the previous chapter, it is possible to observe in the graph shown in Figure D.2 that there are two edges connecting Facebook and Twitter nodes. This can be explained as a problem related to the identification key used to store Facebook and Twitter profiles in the database. The id of every profile in Borges database is equal to the original user id assigned to that profile in the social networking site where they belong, but Facebook and Twitter happen to use a similar id system, so it is possible, although highly unlikely, that two users from different networks have the same identification number assigned to their profiles, which could cause confusion when including these profiles into the database. What seems to be the most efficient way to solve this is the addition of a single character at the beginning of the id number when it is being stored in the database or added to the contacts list of other Borges entries. That character could be a 't' for Twitter connections and a 'l' for Facebook connection, so there would be no confusion between the profiles collected from both social networks.
- The graph view system generates and upload a graph file called "graph.json" to the server, that file will be opened by the JavaScript code to draw the graph image when the mobile device browser accesses to the url path where "index.html" is located. The current system does not make a difference between users accessing to that url, so it will always show the last graph that has been uploaded. This causes a security problem because the system is allowing every one to see the network graph of the last user who uploaded it. To fix this, it is as simple as uploading the graph files using different names for each file, so when the user wants to see the graphic representation of his own network, the application has to open the browser by including in the HTTP request a parameter indicating the name of the graph file for that particular user.

## 7.3 Further Work

The Social Unifier system is meant to be the purely research product of this master thesis, to propose solutions and provide analysis for the issues raised, however, the project has the potential to evolve to reach other magnitudes and other purposes. First of all, besides taking care of the issues described in the section above, and in order to improve the performance of the system, it seems necessary to implement most demanding test plans, to ascertain how the system responds to larger networks and large amounts of information. Having verified that the matching algorithm, in the way it has been raised, shows results that could still be improved, it would be wise to study new ways of extending the

information about the profiles to increase the number of matches found and, in the same way, to improve the ratio of suggested connections for every application user too.

During this thesis it has been intended to develop a system that can be highly scalable to be a part of other systems or the starting point for a whole new other product. Social Unifier provides a simple way to connect to the APIs of the three main social networking sites and to create and manage a CouchDB database. Any kind of social functionality could be built in the system, from synchronizing it with the mobile device address book to give the user the possibility of posting new information in his profiles or sending messages to other users. The idea of improving the usability and the aesthetic design of the Android application, along with the addition of new functionalities that may be attractive to users, would bring to the project the opportunity of handling a huge amount of information to be studied to improve the knowledge about social interactions and the use people give to the social networking platforms.

# Conclusion

---

The problem that has arisen in this master thesis is based on the evolution of new methods of communication and social interactions through the use of smartphones and social networking platforms. The single question that was trying to be answered in this research is the following:

- Is it possible to simplify the three most important social networks by the junction of them into a single network?

This question has been divided into the two goals of this project:

- The creation of a single database that stores a social network by using a graph structure.
- The implementation of an algorithm that finds common points into the social networks and joins them by matching profiles across platforms.

The answer given to this question by this research is the Social Unifier system, which, to a greater or lesser extent, has managed to address their objectives. First of all, Borges database perfectly fulfills the requirements expected for

the database of the system, providing excellent results when testing and being properly adapted to its use as the input to the system algorithms. The biggest challenge of this thesis has been to achieve the desired characteristics of this storing system and the implementation of the writing and reading algorithms that interact with it, which have been carefully design to make the most of the database. Secondly, the test plan has proved that, although the information that can be obtained through the APIs of the social networks is limited and not as extensive as it was desirable, it is still possible to find a significant number of matching profiles across the three of them and this number may increase with further study of the subject. And finally, and most important, Social Unifier lays the foundations for the development of a social networking management software that concentrates all the functionalities of the social platforms into a single tool for smartphones.

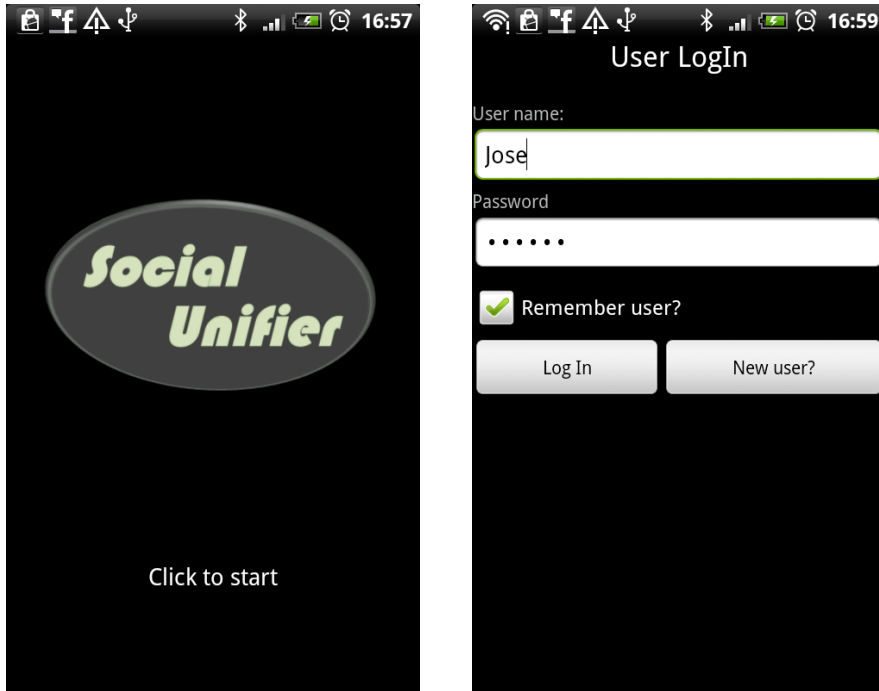


## APPENDIX A

# Screen Captures

---

This appendix includes the screen captures of the Social Unifier application generated by the layouts of the project. Every figure is accompanied by a brief description of the elements included in its layout and the roles they play in the Android application.



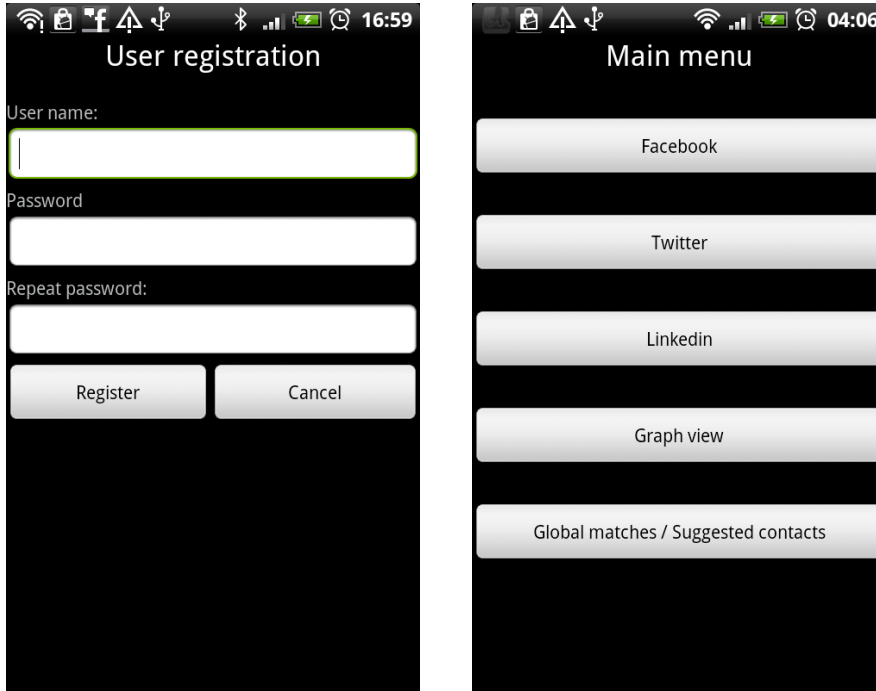
(a) Home screen.

(b) Login screen.

**Figure A.1:** Home and login screens

**Home screen:** The home screen is generated by the file "main.xml". It includes an "ImageView" element, that works as a button to go the the login screen, and a "TextView" element to display the text.

**Login:** The login screen comes from the "login.xml" layout. It contains three "TextView elements" to show the text and two "EditText" elements, one of them to write the user name and the other one to write the password. The second one uses the parameter 'android:inputType="textPassword"' to hide the written characters. This layout also has a "CheckBox" element, for the user to indicate if he wants to keep his user name and the password stored in the device, and two buttons, one of them to login and go to the main menu and the other one to go to the registration screen.



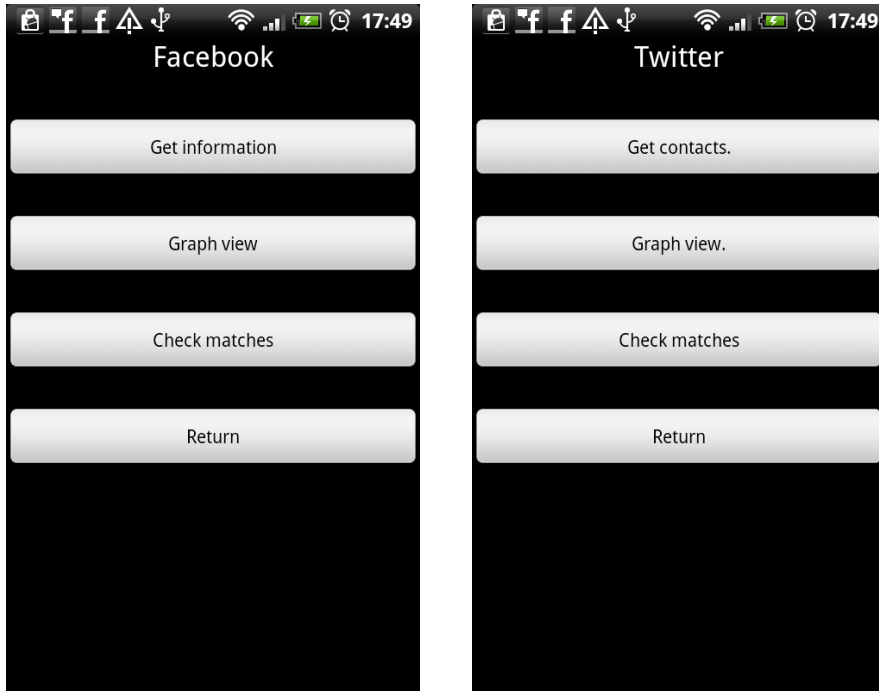
(a) Registration screen.

(b) Main menu screen.

**Figure A.2:** Registration and main menu screens

**Registration:** "register.xml" contains the information about the registration screen, this layout presents the registration form of the system and it contains four "TextView" elements and three "EditText", one for the user name and two for the password and the password confirmation. Finally it has two buttons, one of them to complete the registration and the other one to cancel it, both of them take the user from this layout back to the login screen.

**Main menu:** The main menu screen is generated by the file "menu.xml". It contains one "TextView" element for the title and five buttons. The first three buttons go to the layouts of the Facebook menu, the Twitter menu and the LinkedIn menu. The fourth and the fifth ones open a custom dialog layout that is shown in figures A.5(a) and A.6(a).



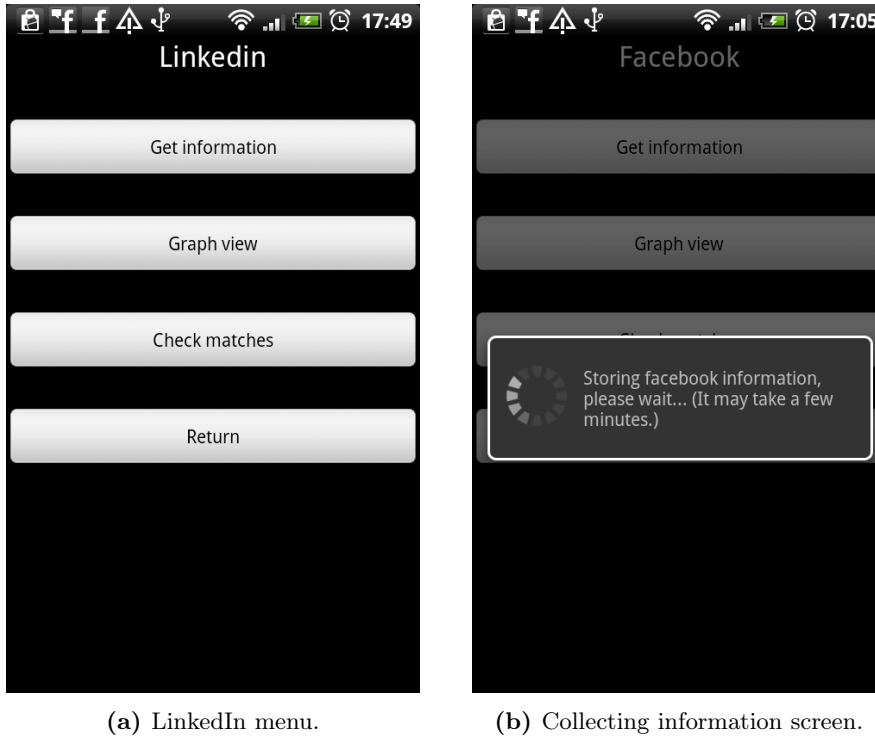
(a) Facebook menu.

(b) Twitter menu.

**Figure A.3:** Facebook menu and Twitter menu screens

**Facebook menu:** "fb\_login.xml" has for buttons an a "TextView" for the tittle. The first button opens a progress dialog and starts the algorithm to get the contacts information, as it can be seen in figure A.4(b). The second and the third button open a custom dialog before going to the graph view screen or the check matches screen. This is shown in figures A.5(a) and A.6(a). The last button simply returns to the main menu screen.

**Twitter menu:** The twitter menu screen is contained in the file "tw\_login.xml" and it has exactly the same elements than the facebook menu and every button has similar behavior too.



**Figure A.4:** LinkedIn menu and collecting information screens

**LinkedIn menu:** generated by "lk\_login.xml", follows the same pattern than the Facebook menu and the Twitter menu.

**Collecting information:** When the "Get information" button is pressed in the LinkedIn, Facebook or Twitter menu, it shows a default progress dialog that displays a message for the user to wait until the information is stored in the database. When this dialog closes the user stays in the same menu screen.

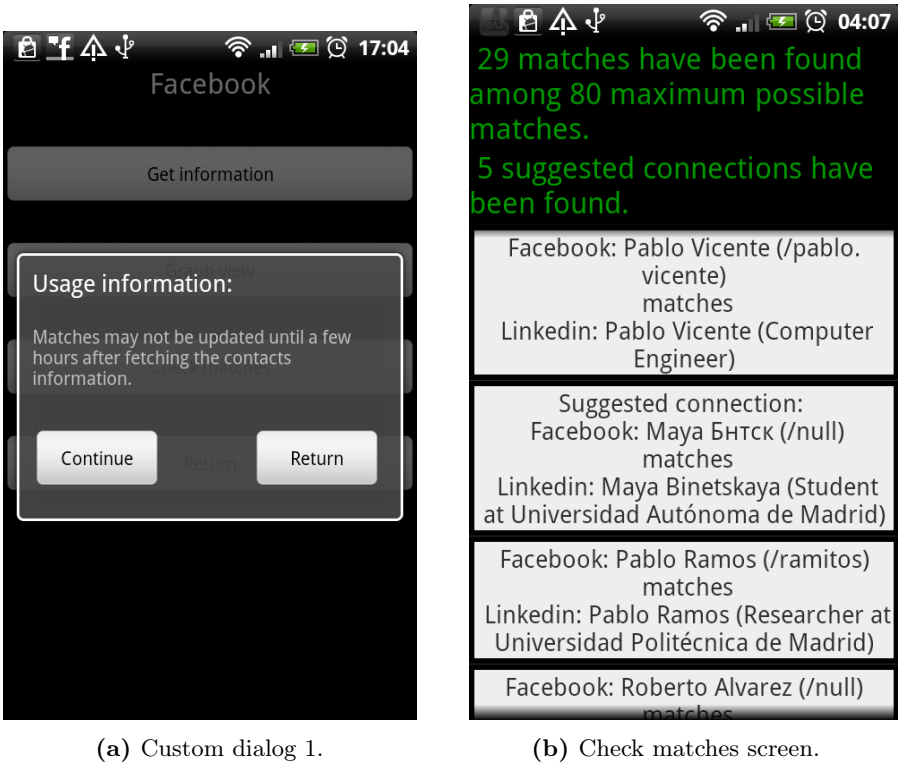
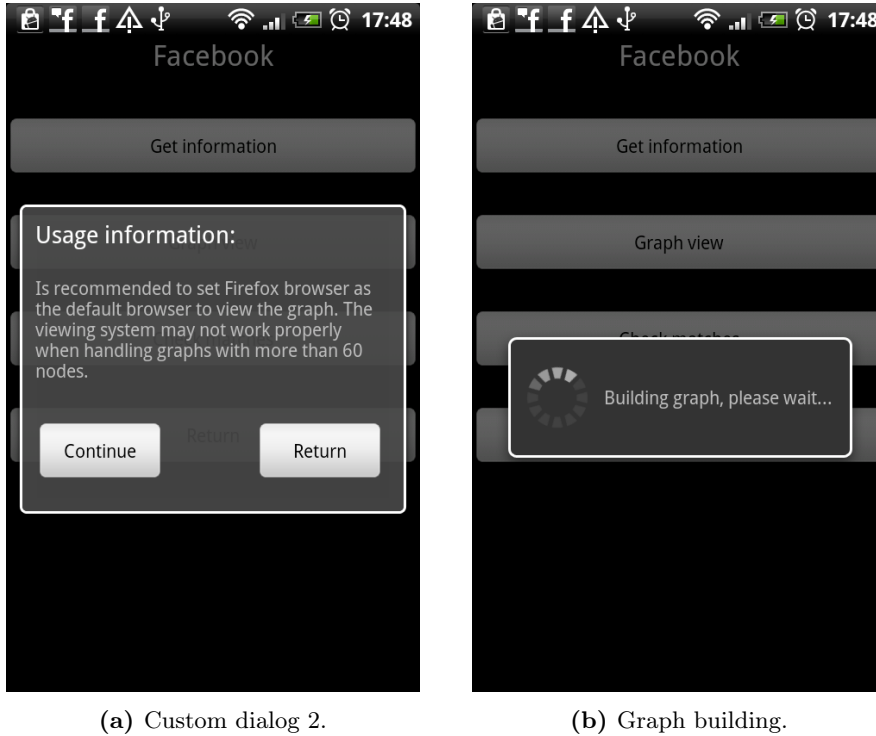


Figure A.5: Custom dialog 1 and "check matches" screens

**Custom dialog 1:** This custom dialog is shown when the user selects to go to the check matches screen from the main menu or the Facebook, LinkedIn or Twitter menu. It is written in the "maindialog.xml" file and it includes a text view with the warning message to be displayed and two buttons, one of them to continue and go to the check matches screen and the other one to cancel the action and go back to the original menu screen.

**Check matches:** The check matches screen is shown when the user selects the option "global matches / suggested connections" from the main menu or "check global matches" from the LinkedIn, Twitter or Facebook menus, and presses the "Continue" button in the custom dialog. The "matches.xml" file contains two "TextView" elements. The first one displays the ratio *matches detected / potential matches* and the second one displays the number of suggested contacts found, but this one will be used only in the case of opening this layout from the "G\_Matches" Java class. There is also a "ListView" element, the elements inside of this "ListView" are generated by "list\_black\_text.xml", which has a "TextView" that will print each one of the detected matches and suggested connections, if necessary. This layout does not have any button, and the user has to press the "return" actual button of the device to go back to the previous screen.



**Figure A.6:** Custom dialog 2 and graph building screens

**Custom dialog 2:** This custom dialog is shown when the user selects the graph view option from the main menu or the Facebook, LinkedIn or Twitter menu. It uses the same layout that the custom dialog 1, "maindialog.xml" and the only difference is in the message displayed. When the user presses the "Continue" button, the program goes to the graph building screen. In case the user presses "Cancel" it goes back to the original menu.

**Graph building:** After the user clicks the "Continue" button in the custom dialog 2, the application stays in the current menu screen and opens a progress dialog, the same way it does when collecting information from the social networks menu. When the graph is built and uploaded to the server this dialog is dismissed and the application opens the default browser to show the requested graph view.





APPENDIX B

# Class Overview

---

There are two packages of Java classes in this project, "thesis.unifier", containing the bulk of the activities and classes of the application, and "org.base", where the class implemented to manage the Borges database is defined. The next sections present the figures that illustrate the overview of all those classes.

## B.1 Social Unifier Activities

The Social Unifier Android manifest document declares the existence of twelve activities. The classes that extend the class "Activity" are contained in the "thesis.unifier" package and the figures illustrating their classes overview are shown below.

Main
img : ImageView
onCreate(savedInstanceState : Bundle) : void onActivityResult(requestCode : int,resultCode : int,data : Intent) : void

Figure B.1: Main class overview.

Login
checkBox : CheckBox t_user : EditText t_pass : EditText bLogin : Button bReg : Button save : boolean pass : String name : String settings : SharedPreferences editor : SharedPreferences.Editor db : Database user : JSONObject
onCreate(savedInstanceState : Bundle) : void onActivityResult(requestCode : int,resultCode : int,data : Intent) : void

Figure B.2: Login class overview.

Register
t_user : EditText t_pass : EditText t_pass1 : EditText pass : String pass1 : String name : String bLogin : Button bCancel : Button db : Database json : JSONObject
onCreate(savedInstanceState : Bundle) : void onActivityResult(requestCode : int,resultCode : int,data : Intent) : void

Figure B.3: Register class overview.

Menu
bbtnf : Button bbtnnt : Button bbtnl : Button bbtnng : Button bbtnnm : Button settings : SharedPreferences user_name : String dialog : dialog
onCreate(savedInstanceState : Bundle) : void onActivityResult(requestCode : int,resultCode : int,data : Intent) : void

Figure B.4: Menu class overview.

Fb_main
facebook : Facebook mPrefs : SharedPreferences btn1 : Button btn2 : Button btn3 : Button btn4 : Button db : Database response : String response2 : String settings : SharedPreferences user_name : String main_doc : JSONObject doc_aux : JSONObject fb_users : JSONObject doc_list : String aux : String list_aux : String list_aux2 : ArrayList<String> dialog : Dialog
onCreate(savedInstanceState : Bundle) : void onActivityResult(requestCode : int,resultCode : int,data : Intent) : void

Figure B.5: Fb\_main class overview.

Tw_main
prefs : SharedPreferences setting : SharedPreferences user_name : String db : Database main_doc : JSONObject tw_users : JSONObject msg : String flag : int dialog : Dialog
onCreate(savedInstanceState : Bundle) : void

Figure B.6: Tw\_main class overview.

Lk_main
btn1 : Button btn2 : Button btn3 : Button btn4 : Button settings : SharedPreferences user_name : String db : Database ld_users : JSONObject dialog : Dialog oAuthService : LinkedInOAuthService factory : LinkedInApiClientFactory liToken : LinkedInRequestToken client : LinkedInApiClient
onCreate(savedInstanceState : Bundle) : void startAuthenticate() : void finishAuthenticate(uri : Uri) : void clearTokens() : void getData(accessToken : LinkedInAccessToken) : void onNewIntent(intent : Intent) : void

Figure B.7: Lk\_main class overview.

G_Matches
doc : JSONObject user : JSONObject db : Database settings : SharedPreferences user_name : String counter : int err_c : int
onCreate(savedInstanceState : Bundle) : void

Figure B.8: G\_Matches class overview.

FB_Matches
doc : JSONObject user : JSONObject db : Database settings : SharedPreferences user_name : String counter : int err_c : int
onCreate(savedInstanceState : Bundle) : void

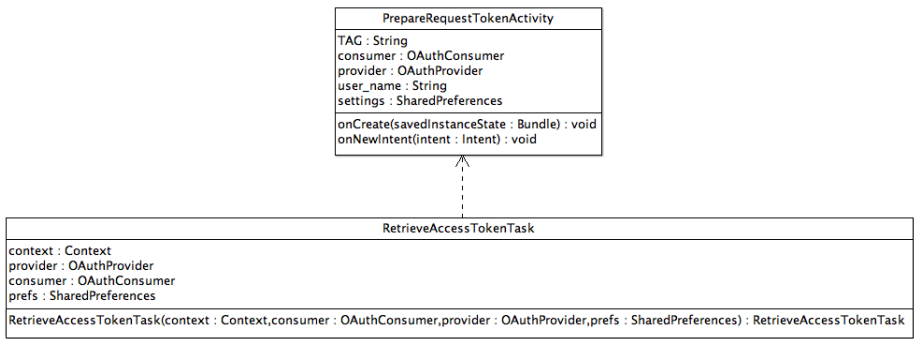
Figure B.9: FB\_Matches class overview.

TW_Matches
doc : JSONObject user : JSONObject db : Database settings : SharedPreferences user_name : String counter : int err_c : int
onCreate(savedInstanceState : Bundle) : void

Figure B.10: TW\_Matches class overview.

LD_Matches
doc : JSONObject user : JSONObject db : Database settings : SharedPreferences user_name : String counter : int err_c : int
onCreate(savedInstanceState : Bundle) : void

Figure B.11: LD\_Matches class overview.



**Figure B.12:** PrepareRequestTokenActivity and RetrieveAccessTokenTask classes overview. RetrieveAccessTokenTask is declared inside of PrepareRequestTokenActivity.

B.2 No Activity Classes

Class overviews of the rest of the classes that are part of the "thesis.unifier" package are shown below.

Constants
PASSWORD : String
USERNAME : String
TW_CONSUMER_KEY : String
TW_CONSUMER_SECRET : String
TW_REQUEST_URL : String
TW_ACCESS_URL : String
TW_AUTHORIZE_URL : String
TW_OAUTH_CALLBACK_SCHEME : String
TW_OAUTH_CALLBACK_HOST : String
TW_OAUTH_CALLBACK_URL : String
LD_OAUTH_CALLBACK_SCHEME : String
LD_OAUTH_CALLBACK_HOST : String
LD_OAUTH_CALLBACK_URL : String
PREFS_NAME : String
LD_CONSUMERKEYVALUE : String
LD_CONSUMERSECRETVALUE : String
LD_OAUTH_PREF : String
LD_PREF_TOKEN : String
LD_PREF_TOKENSECRET : String
LD_PREF_REQTOKENSECRET : String
LD_OAUTH_QUERY_TOKEN : String
LD_OAUTH_QUERY_VERIFIER : String
LD_OAUTH_QUERY_PROBLEM : String
FB : int
LD : int
TW : int
ALL : int

**Figure B.13:** Constants class overview.

TwitterUtils
<pre>isAuthenticated(prefs : SharedPreferences) : boolean getInfo(prefs : SharedPreferences) : User getUsersInfo(prefs : SharedPreferences,ids : long[]) : ResponseList&lt;User&gt; getFriends(prefs : SharedPreferences,id : long) : ArrayList&lt;Long&gt; getFriends(prefs : SharedPreferences,id : long,l_friends : ArrayList&lt;Long&gt;) : ArrayList&lt;Long&gt;</pre>

Figure B.14: TwitterUtils class overview.

OAuthRequestTokenTask
<pre>TAG : String context : Context provider : OAuthProvider consumer : OAuthConsumer  OAuthRequestTokenTask(context : Context,consumer : OAuthConsumer,provider : OAuthProvider) : OAuthRequestTokenTask doInBackground() : void</pre>

Figure B.15: OAuthRequestTokenTask class overview.

Graph
<pre>g : JSONObject counter : HashMap db : Database  generateGraph(user_name : String,social : int) : JSONObject</pre>

Figure B.16: Graph class overview.

Upload
<pre>up(user_name : String,social : int) : void</pre>

Figure B.17: Upload class overview.

### B.3 Database Class

The package "org.base" contains the Database class, which provides the tools to connect and make requests to the Borges database

Database
url : String user : String password : String
Database(url : String) : Database Database(url : String,user : String,password : String) : Database read(request_doc : String) : JSONObject write(doc : JSONObject,key : String) : JSONObject update(doc : JSONObject,key : String) : JSONObject delete(key : String) : boolean

**Figure B.16:** Database class overview.





## APPENDIX C

# Network Plot Method

---

Appendix D includes figures that show the image of Social Unifier networks. To plot this social network data it has been used the Gephi [26] open source software, which draws the graph after importing the data from a ".dot" [27] file generated by a Python script. This python script uses the "couchdb" module to read the database, and the "networkx" [28] package to build the graph from the information stored in Borges. Once the graph is built, the script writes the ".dot" file by using the method *nx.drawing.write\_dot(G, file\_path)*, for which it is necessary to install the "Graphviz" [29] graph visualization software and "PyGraphviz" [30], the Python interface to the Graphviz graph layout and visualization package. The following section presents the code of the script.

### C.1 GraphGenerator.py

```
import sys
import os
import couchdb
import networkx as nx

try:
    import jsonlib2 as json
```

```

except ImportError:
    import json

## Include Database credentials.
username = "s081289"
password = "socialunifier"
db = couchdb.Database("http://lestrade.imm.dtu.dk:5984/s081289_borges")
db.resource.credentials = (username, password)

G=nx.Graph()

fb = db['fb_users']
lk = db['ld_users']
tw = db['tw_users']
matches = db['matches']

for doc in db:
    if doc != 'fb_users' and doc != 'ld_users' and doc != 'tw_users' and doc != 'matches':
        entry = db[doc]
        if 'linkedin_info' in entry:
            if entry['linkedin_info']['id'] in lk:
                name = lk[entry['linkedin_info']['id']]
                G.add_node(name, group='user')
            else:
                name = db[doc]['linkedin_info']['first_name'] +
db[doc]['linkedin_info']['last_name']
                if entry['linkedin_info']['id'] in matches:
                    G.add_node(name, group='match')
                else:
                    G.add_node(name, group='linkedin')
        elif 'facebook_info' in entry:
            if entry['facebook_info']['id'] in fb:
                name = fb[entry['facebook_info']['id']]
                G.add_node(name, group='user')
            else:
                name = db[doc]['facebook_info']['name']
                if entry['facebook_info']['id'] in matches:
                    G.add_node(name, group='match')
                else:
                    G.add_node(name, group='facebook')
        elif 'twitter_info' in entry:
            if entry['twitter_info']['id'] in tw:
                name = tw[entry['twitter_info']['id']]
                G.add_node(name, group='user')
            else:
                name = db[doc]['twitter_info']['name']
                if str(entry['twitter_info']['id']) in matches:

```

```

        G.add_node(name, group='match')
    else:
        G.add_node(name, group='twitter')
else:
    name = doc
    G.add_node(name)
if 'twitter_contacts' in db[doc]:
    con = db[doc]['twitter_contacts']
    if con != "[]":
        list = db[doc]['twitter_contacts'][1:-1].split(', ')
        for x in list:
            if x in tw:
                G.add_edge(name,db['tw_users'][x])
            elif x in matches:
                m = matches[x]
                if len(matches[x]) == 1:
                    match = db[matches[x][0]]
                    if 'linkedin_info' in match:
                        name2 = match['linkedin_info']['first_name'] +
match['linkedin_info']['last_name']
                        G.add_edge(name,name2)
                    else:
                        G.add_edge(name,match['facebook_info']['name'])
                else:
                    match = db[matches[x][0]]
                    if 'linkedin_info' in match:
                        name2 = match['linkedin_info']['first_name'] +
match['linkedin_info']['last_name']
                        G.add_edge(name,name2)
                    else:
                        d = db[matches[x][1]]
                        name2 = d['linkedin_info']['first_name'] +
d['linkedin_info']['last_name']
                        G.add_edge(name, name2 )
            else:
                G.add_edge(name,db[x]['twitter_info']['name'])

if 'facebook_contacts' in db[doc]:
    con = db[doc]['facebook_contacts']
    if con != "[]":
        list = con[1:-1].split(', ')
        for x in list:
            if x in fb:
                G.add_edge(name,db['tw_users'][x])
            elif x in matches:
                m = matches[x]
                if len(matches[x]) == 1:

```

---

```

        match = db[matches[x][0]]
        if 'linkedin_info' in match:
            name2 = match['linkedin_info']['first_name'] +
match['linkedin_info']['last_name']
            G.add_edge(name,name2)
        else:
            G.add_edge(name,db[x]['facebook_info']['name'])
    else:
        match = db[matches[x][0]]
        if 'linkedin_info' in match:
            name2 = match['linkedin_info']['first_name'] +
match['linkedin_info']['last_name']
            G.add_edge(name,name2)
        else:
            d = db[matches[x][1]]
            name2 = d['linkedin_info']['first_name'] +
d['linkedin_info']['last_name']
            G.add_edge(name,name2 )
    else:
        G.add_edge(name,db[x]['facebook_info']['name'])

    if 'linkedin_contacts' in db[doc]:
        con = db[doc]['linkedin_contacts']
        if con != "[]":
            list = con[1:-1].split(', ')
            for x in list:
                if x in lk:
                    G.add_edge(name,db['ld_users'][x])
                else:
                    name2 = db[x]['linkedin_info']['first_name'] +
db[x]['linkedin_info']['last_name']
                    G.add_edge(name,name2)

# Write graph file.
nx.drawing.write_dot(G, "out.dot")

```

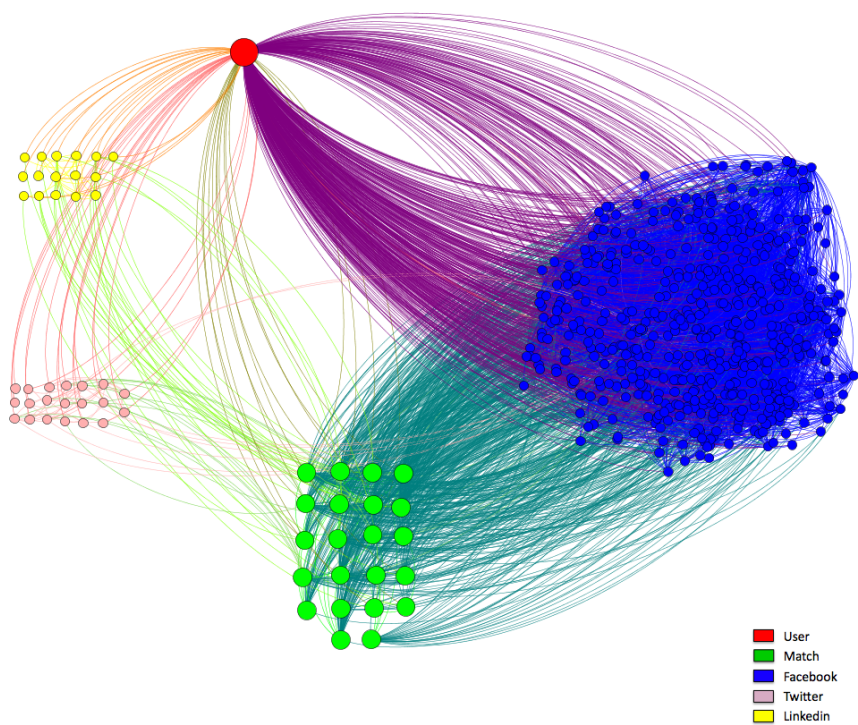
## APPENDIX D

# Social Unifier Network Graphics

---

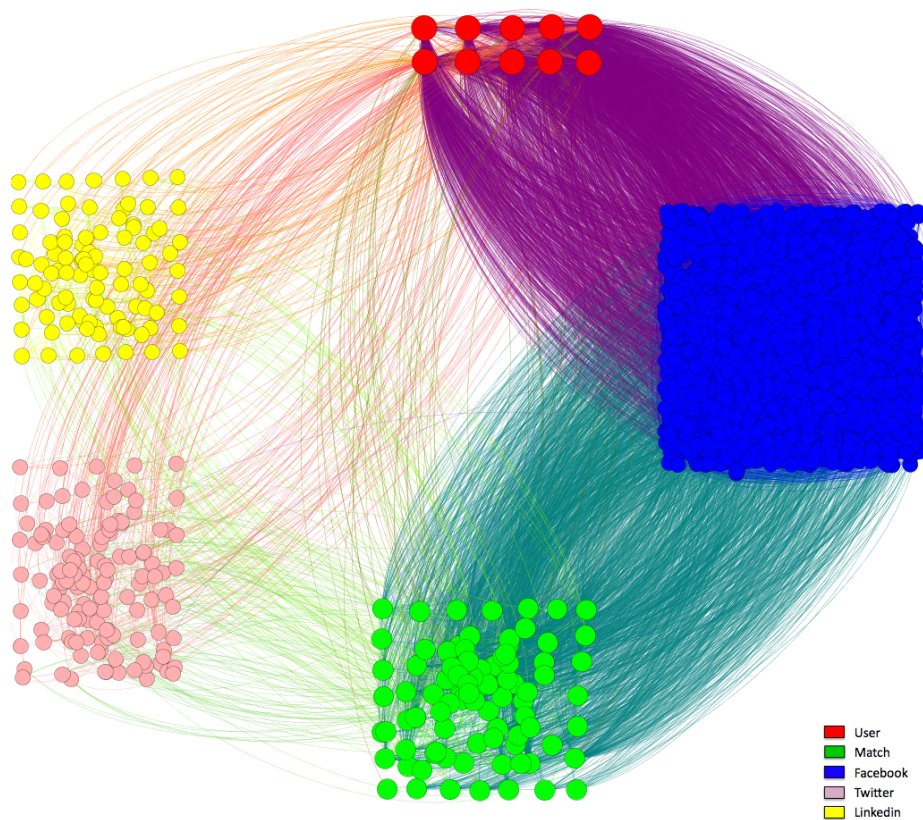
The next two sections contain figures D.1 and D.2, which show the state of the Social Unifier network for a single user of the system and the image of the global network after the test plan has been conducted. The purpose of this images is to give an idea of how the contacts from the three social networks relate to each other and how the networks connect between them through the found matching profiles. The figure D.2 shows a graph that can be understood as the junction of ten graphs that are similar to the one displayed in the figure D.2.

D.1 Single User Network



**Figure D.1:** Network of the Social Unifier test User 1.

## D.2 Global Network



**Figure D.1:** Social Unifier Network after the test plan has been conducted.





# Bibliography

---

- [1] EbizMBA, *Top 15 Most Popular Social Networking Sites*, June 2012, <http://www.ebizmba.com/articles/social-networking-websites> .
- [2] Alexa Top Sites list, <http://www.alexa.com/topsites> .
- [3] J. Chris Anderson, Jan Lehnardt & Noah Slater. *CouchDB: The definitive guide*. O'reilly, 1st ed. January 2010. <http://guide.couchdb.org/> .
- [4] Python-twitter, a Python wrapper around the Twitter API, <http://code.google.com/p/python-twitter/> .
- [5] Twitter developers documentation, <https://dev.twitter.com/docs> .
- [6] Redis, <http://redis.io/> .
- [7] Facebook-Python SDK, <http://pypi.python.org/pypi/facebook-python-sdk> .
- [8] Facebook developers Graph API, <https://developers.facebook.com/docs/reference/api/> .
- [9] Facebook Query Language (FQL), <https://developers.facebook.com/docs/reference/fql/> .
- [10] Python LinkedIn Module, a python wrapper around the LinkedIn API, <http://code.google.com/p/python-linkedin/> .
- [11] LinkedIn developers APIs, <https://developer.linkedin.com/apis> .
- [12] CouchDB-Python Library, <http://code.google.com/p/couchdb-python/> .

- [13] CouchBase, <http://www.couchbase.com/>
- [14] Apache CouchDb, <http://couchdb.apache.org/>
- [15] Data-Driven Documents, D3.js, [Data-DrivenDocuments](http://Data-DrivenDocuments.com).
- [16] JCraft Java Secure Channel (JSch), <http://www.jcraft.com/jsch/>.
- [17] Matthew A. Russell, *Mining the Social Web*. O'reilly, 1st ed. January 2011. Pages 84-87. <http://shop.oreilly.com/product/0636920010203.do>
- [18] Eclipse IDE for Java Developers, <http://www.eclipse.org/downloads/moreinfo/java.php>.
- [19] Android Developers, Android SDK, <http://developer.android.com/sdk/index.html>.
- [20] Android Developers, Android 2.2 Platform, <http://developer.android.com/sdk/android-2.2.html>
- [21] Python 2.7.2 release, <http://www.python.org/download/releases/2.7.2/>
- [22] The Official Facebook SDK for Android, <https://github.com/facebook/facebook-android-sdk/>
- [23] OAuth-Signpost, Simple OAuth message signing for Java, <http://code.google.com/p/oauth-signpost/>
- [24] Twitter4j, a Java library for the Twitter API <http://twitter4j.org/en/index.html>
- [25] LinkedIn-j, a Java wrapper for LinkedIn APIs, <http://code.google.com/p/linkedin-j/>.
- [26] Gephi, an open source graph visualization and manipulation software, <https://gephi.org>.
- [27] The DOT language <http://www.graphviz.org/content/dot-language>.
- [28] NetworkX, high productivity software for complex networks, <http://networkx.lanl.gov/>.
- [29] Graphviz, graph visualization software, <http://www.graphviz.org>.
- [30] Pygraphviz, Python interface to the Graphviz graph layout and visualization package, <http://networkx.lanl.gov/pygraphviz>.