Vrije Universiteit Brussel

FACULTY OF ENGINEERING
Department of Computer Science - WISE
Master Applied Science and Engineering: Computer Science

A Context-Aware View Over Social Media

Graduation thesis submitted in partial fulfillment of the requirements
for the degree of Master Applied Science and Engineering: Computer Science

Luong Dinh Hieu

Promoter: Prof. Dr. Olga De Troyer
Advisor: Dr. William Van Woensel, Prof. Dr. Sven Casteleyn

2013-2014

Acknowledgements

First of all, I would like to thank my promoter Prof. Dr. Olga De Troyer for her valuable advices, support and encouragement.

I would like to thank BTC organization for their financial support. I also want to thank Vrije Universiteit Brussels (VUB) for accepting me as a student and VUB's staff for their help when I arrived at the university.

I would also like to thank William Van Woensel - my advisor for his great patience, enthusiasm, and encouragement during the time of doing this thesis. His vast experience in doing researched helped me a lot in my thesis work. Without him, this thesis would never been accomplished.

Another advisor, Prof. Dr. Sven Casteleyn, also deserves my gratefulness for his effort in correcting this dissertation. His corrections and advices help me so much during the accomplishment of this thesis.

Finally, I thank my friends and my family who helped me and encourage me spiritually during the time I was doing the thesis.

Abstract

The capabilities of mobile devices have increased with leaps and bounds. Nowadays, with a smartphone in hand, mobile users can run powerful mobile applications (e.g., office apps, games, route planners) anywhere and anytime. In addition to increased memory and processing power, mobile devices are also being outfitted with a range of detection technologies (e.g., NFC, GPS). Importantly, these technologies enable the collection of the mobile user's context. For example, at a short distance, technologies such as Bluetooth, RFID and NFC may discover surrounding objects, including people, places and things. In addition, GPS technology allows us to determine the location of user, and built-in accelerometers and digital compasses identify human motion and orientation. When equipped with these technologies, mobile devices are able to sketch an overall contextual picture about the mobile user.

At the same time, the explosive growth of social media allows people to connect each other in the online community. A wide variety of social media platforms exist to suit all sorts of social needs, such as communication in general (e.g., Facebook, Twitter), business and professional networking (LinkedIn), education (StudiVZ), music (Last.fm), and so on. In line with the current popularity of mobile devices, social media is being more and more deployed on mobile platforms. This includes mobile apps to supply instant and real-time access to social media (e.g., Twitter, Facebook), and directly share photos and videos online with family and friends (e.g., Instagram). Due to this mobile deployment, an excellent opportunity exists to automatically enhance social media interaction by leveraging captured mobile user context. Current examples include the geo-tagging of pictures shared on a social media platform (e.g., Facebook, Google Plus, etc.), and automatically discover services based on the user's location (e.g., Foursquare).

In particular, an important opportunity exists to realize a full, bi-directional integration of social media with the physical world. In one direction, context collected by mobile devices could be utilized to automatically enrich social media content. For example, information of a new place that users visited could be shared to the online community automatically. This automation reduces the burden of manual user's context annotation. In another direction, the relevant and useful information can be automatically retrieved from social media, by leveraging the mobile user's context. For example, when travelling to a new place, any information related to that place obtained from social media (e.g., the user's friends have been there, or information of some event is occurring at that place) could be interesting or helpful to the user.

The goal of this thesis is to investigate and develop a generic software framework, realizing a full bi-directional integration of social media and real world activities. Furthermore, it allows integrating any social network, and thus provides us an opportunity to exploit context-relevant information from different sources at once. In addition, integrating multiple social networks is necessary because people are typically subscribed to different social network platforms for different purposes. Finally, an android application is built upon this framework. This is a proof-of-concept application with a user-friendly mobile interface, grouped around important social media concepts.

Contents

List Of Figures

1 Introduction

This chapter presents the research context and problem statement, as well as a summary of the approach and the structure of this dissertation. In particular, the research context indicates the motivations and opportunities of integrating social media on mobile.

1.1 Research Context

Online social media has become incredibly popular, and is being accessed by a wide range of different people for different purposes (e.g., LinkedIn for business, Facebook for communication, StudiVZ for education). In 2012, the use of social media reached 1 billion users and in the last year (2013) the growth shows no signs of slowing down (see figure 1). Social media has been deeply integrated into our routine and is having an increasing impact on human society. People use social media as a new way of communication and sharing information. In business, social networks are used as a tool for product advertisement.

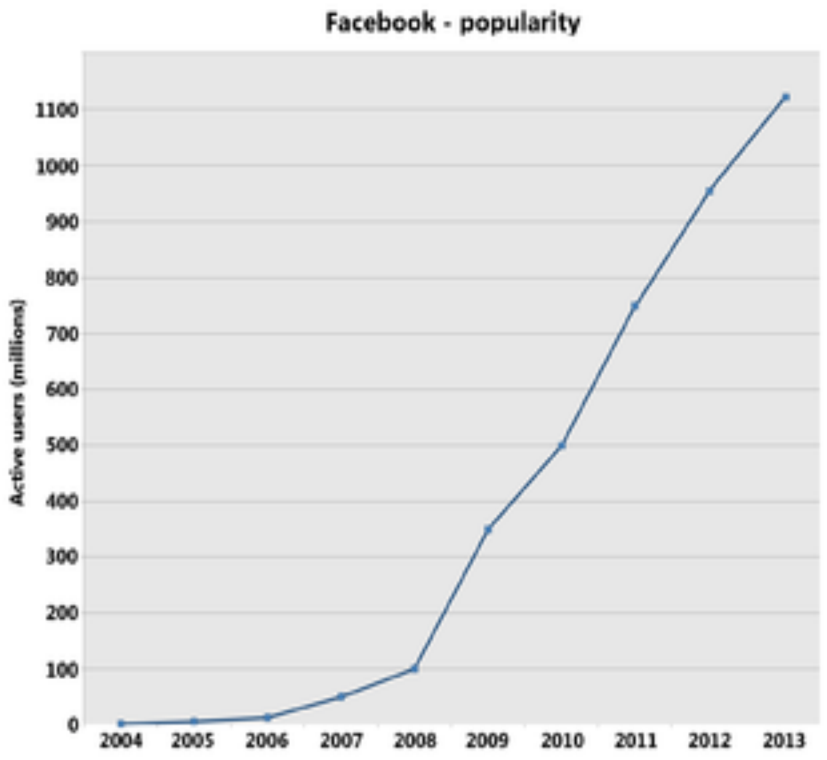


Figure 1 - Facebook popularity. Active users of Facebook increased from just a million in 2004 to over 750 million in 2011. Taken from Wiki source [1]

[1] http://en.wikipedia.org/wiki/Facebook

Parallel to the widespread usage of social media, we observe the rapid growth of the smartphone and tablet markets. Combining these two evolutions, people are getting more involved in social networking and on advanced mobile devices. Smartphone users are spending more time on social networks - say, posting status updates on Facebook, tweeting their thoughts to followers on Twitter or checking-in on Foursquare. Up to February 2013, according to the MarketingCharts Mobile Consumer Survey[2], 55% of social networking consumption occurs on mobile devices. There are also numerous applications, allowing users to access social media platforms (Facebook, Twitter, Tumblr, Google Plus, etc.) on mobile devices. By deploying social media on mobile devices, mobile users are empowered to directly post updates related to their changing locations (e.g., having a nice lunch at the VUB KultuurKaffee), or directly tag location to their photos (e.g., a photo with description "Light Festival at Amateur Square"); and vice-versa, finding social media information related to their current surroundings (e.g., friend's reviews of the KultuurKaffee, or future events attended by friends at that place). In doing so, mobile social media platforms allow for a strong intertwinement of social media real life.

Advanced sensing capabilities of smartphones and tablets allow identifying the mobile user's current location and surroundings. For instance, built-in cameras (to read Quick Response codes), as well as RFID (Radio Frequency Identification) and NFC (Near Field Communication) readers not only allow integration with nearby tags and devices, but collecting data from them as well. The mobile user's current, fine-grained location is retrieved via GPS technology, and compasses and accelerometer acquire human motion states. As a result, the availability of such sensing hardware opens an excellent opportunity to obtain the mobile user's physical context.

The deployment of social media platforms (Facebook, Google Plus, etc.) on mobile devices, together with the context-sensing capabilities of current mobile devices, result in opportunities to automatically integrate social media with activities in the physical world. Currently, such integration is already being performed to an extent. Efforts are underway by social media providers themselves (e.g., almost social medias support geo-tagging feature, letting users store or keep track of data based on location; Foursquare helps smartphone users find the perfect places in their vicinity, such as discovering best coffee shops, restaurants, coupons or promotion). Moreover, such integration is also investigated in related researches. This includes automatic status updates, based on inferred daily activities (IYOUIT [11]). This also includes automatic extracting context-relevant social media data and thereby giving recommendations based on the user's situation (WhozThat [2], SocialFlick [3]). All of these efforts reflect the importance of integrating social media with activities in the physical world.

[2] http://www.marketingcharts.com/wp/interactive/55-of-social-networking-consumption-occurs-on-a-mobile-device-27327/

1.2 Problem Statement

As mentioned, social media platforms are increasingly dealing with the integration of social media and physical activities. In related works, this integration is currently done even more extensively. However current approaches performing this integration are not very far-reaching in the two following fundamental ways:

Firstly, many people are members of multiple social media platforms, but current approaches only consider a single or few platforms at a time. Nowadays, joining multiple social medias is normal practice. For example, Facebook is now part of most people's lives; Twitter is where a lot of people are reading the breaking news. YouTube is a place for people sharing and watching videos. LinkedIn helps you in your own professional career. Consequently, exploiting context-relevant information from multiple sources can be useful to leverage the user's context. By integrating more social networks, more useful social media data becomes available. However, previous works are limited to a single or few social media platforms (see chapter 3).

Secondly, no single social media approach realizes an automatic, fully bi-directional integration with the real world. We define such a bi-directional integration. In one direction, personalized and context-relevant information is automatically extracted from social media platforms for the user. Often, the mobile user is curious or keen on finding context-related social media data; e.g., finding friends attending nearby events, finding any information friends talk about a specific nearby tourist attraction, or discovering photos that friends took at that place. In the other direction, user's context is utilized automatically to enrich the social media content. For example, people tend to share their check-ins actions (the action specifies that a person has visited a particular place). These updates can be done in an automatic way with the support of context-aware mobile applications together with the mobile sensing capabilities, and thus reduce the burden of manual user's context annotation.

In summary, the goal of this thesis is realizing an improved integration of social media and physical world activities by solving two limitations mentioned above. This thesis emphasizes on providing a framework that realizes the fully bi-directional integration, and allows integrating multiple social media platforms at once.

1.3 Goals and Approach

This section describes the targeted results, challenges and how the approach used to obtain the targeted results and deal with the challenges.

1.3.1 Targeted Results

We focus on the two following points:

First, we will define a framework that can realize the fully bi-directional integration with the real life activities. In particular, we seek for an approach that can automatically enhance the online social media content with information from the user's context, and in the opposite way, can offer social media information to the user, taking into account his or her current context.

Secondly, we will propose an approach to support any social media platform.

1.3.2 Challenges

First, while performing the bi-directional integration, the challenge we could meet is how to gather, manage and serve the context information.

Secondly, to ensure the integration of multiple social media platforms at once, we have to deal with various social media data types, and integrate data from various social media. Another challenge is related to the fact that the amount of social media data is huge, so we need a mechanism to determine and rank the context-relevant information obtained from these social media.

1.3.3 Approach

First of all, the software application will employ an existing generic context repository that allows integrating any context acquisition system (e.g., SCOUT [22]) to gather, manage context data (e.g., nearby buildings, monuments, places, persons). The work presented in this dissertation will realize a full integration of social media and activities in the physical world, in both directions, i.e. from the physical world to social media, and from social media to physical world. Below, we elaborate on how both integration directions will be realized in our framework.

Physical world → Social media: This integration means that the user's context information can be obtained and shared on social networks (e.g., context information like current country when the user is abroad, monuments he is visiting, restaurants he had dinner at, etc.). In our approach, these collected context information can be (semi-) automatically posted on online communities.

The auto-posting mechanism is a mechanism that depending on user preferences the messages generated (or deducted) from the user's context can be posted automatically. In our framework, this auto-posting mechanism is performed in the following cases: whenever you go abroad, the country name will be posted, and when the time that you visit a place is longer than a specific amount of hour (e.g., one or two hours) the name of the place will be posted.

In the semi-auto posting mechanism, posting these messages requires user confirmation. In our framework, all contexts will be collected in a history, and at a specific time (i.e. at the end of day) the user could consider if any context information can be posted. Both mechanisms reduce the burden of context annotation, but the semi-auto posting provides a consideration for the user before posting context information online.

Social media → Physical world: Here, context-relevant social media data is automatically pushed towards the user. Based on the information of the user's current location, our framework will access to different social media platforms via their API to obtain the context-relevant data. In the scope of this thesis, we focus on 6 kinds of context relevant data:

- **<u>People</u>**: finding friends whose activities are related to the user's current location (e.g., living, working, studying nearby);

- **<u>Check-ins</u>**: finding friends who visited places in the user's surroundings;

- **<u>Groups</u>**: finding groups with activities that are related to the user's location, such as the group belongs to a particular place, or organizations;

- **<u>Events</u>**: finding events that belongs to a specific entity, such as building, company, university that are located in the user's surroundings, or events that take place nearby user's current location;

- **<u>Posts</u>**: finding all public posts that contains information about a specific entity, such as tourist attraction, organization, university that are located in the user's surroundings;

- **<u>Photos</u>**: finding all photos and albums that friends have taken of places or things in the user's current surroundings.

Secondly, the challenges concerning plugging in any social media platform, we will solve as follow:

As a result of supporting multiple social media platforms, a first challenge is concerned with dealing with different social media data types. The data type structure varies among different social media platforms, by different structures, and different field names. To reconcile the overlaps or conflicts between such data types, we created a common, shared vocabulary defining data types from heterogeneous social networks.

A second challenge is about filtering context-relevant information from the huge set of online social media data. Compounding matters, much of this information is unstructured (e.g., posted messages), making the filtering process more difficult. The approach we propose is applying context-matching function, which find the most relevant social network data based on comparing properties between user's context data and social network data.

Finally, this software framework was implemented for the Android platform. On top of this generic software framework, we implemented a user-friendly, proof-of-concept Android user interface, centered around general concepts occurring in social media (e.g., posts, groups, events, etc.)

1.4 Structure of the Thesis

The remainder of this thesis is structured in the following way:

- The second chapter introduces the underlying concepts of this thesis. At the beginning, the main concepts, user context and context awareness will be explained. Afterwards, we explain how to access a resource on a social network platform. The SCOUT framework that we use to obtain the context data will be studied at the end of this section together with the Semantic Web Technologies (e.g., OWL, RDF, SPARQL, etc.).

- Chapter three discusses and compares related work.

- Chapter four presents our approach.

- Chapter five provides our implementation of building a mobile application upon the framework.

- Chapter six draws conclusions and presents some future works.

2 Background

This chapter provides the background knowledge that we used in the thesis. At the beginning, the terms user context and context awareness will be explained. With the purpose to integrate multiple social networks into the application, investigating how to connect to social networks in general is paramount. We are going to study the common mechanism of how third-party sites are able to access, and extend the social network data. Afterward, we will delve into details of each social network connect service, like Facebook, Google Plus, and Twitter.

2.1 User Context

Before going into the detail of our topic, we need to define what we mean by context. Schilit [20] defined the user context as the combination of three aspects: "where you are, whom you are with, and what resources are nearby". Schilit defined the context as a set of locations, nearby people, buildings and devices. Afterward, many other and similar definitions were given [5], [7], in which the context is defined by enumerating all context types. However, as mention in [6] these types are too specific, and these definitions are quite hard to extend. If a new type of context occurs and is not in the list, then the question is how we can handle it to adapt to the existing system with a fixed set of context types?

Therefore, Dey [6] provides the following definition: "Context is any information that can be used to characterize the situation of an entity. An entity is a person, place, or object that is considered relevant to the interaction between a user and an application, including the user and application themselves". With this definition, the context is extended to anything that can be used to characterize an entity's situation. For example, a person is walking on a street, listening to his favorite song, and reading a book on mobile. On the way, he meets some friends. There are several supermarkets and a famous coffee shop nearby. So, all the activities he is engaged in, nearby locations, nearby people are his contexts. Even, the user's mood or feeling, and the weather, temperature, humidity, etc. could be included in the context.

2.2 Context Awareness

As defined in [6], a system is Context Awareness if it uses context to provide relevant information and/or services to the user, where relevancy depends on the user's task. Let's take a look at the previous example again. On the way, he passes by a famous coffee shop, which is highly recommended or rated by his friends and colleagues on some social networks. Based on his preference mentioning that coffee is one of his favors, the computing mobile will give him a recommendation to this shop together with the friends' reviews.

Context Awareness has become a popular topic for recent years. Its purpose is in making human daily life easier by providing some cues inferred from the user context, such as recommendations for nearby shops, restaurants, means of transportation, and so on.

2.3 Social Networks Connect Service

In Social Network Websites, users can develop their social relationships by sharing individual information on their profile, like status, videos, photos, and messages. However, updating interesting contents may happen when users interact with other applications. For example, you play a game in an application and achieve a really high score. Now, you want to share this high score on Facebook so that your friends can see it and admire you. To encourage this integration, many social networks have exposed their networks to Web services in the form of online application programming interfaces, which brings up benefit for both sides. On the third-party side, these APIs let the developers develop their applications without having to either host or build their own social network, and advertise their services over the social network to gather a large number of users. On the social networks side, the integration provides a richer content, and makes users more involved into their social networks environment. Now, let's take a look at the common mechanism of how third-party sites are able to access, and extend the social network data.

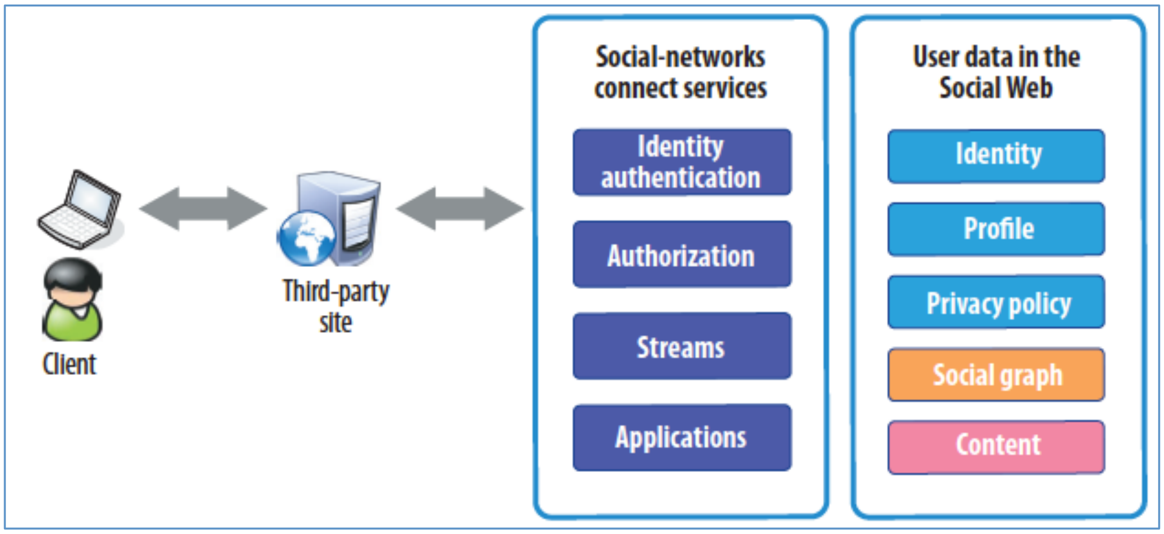


Figure 2 - Social-networks connect services Framework. Taken from [13]

As described quite well in [13], there are four stages for a third-party application to be able to access to a social network: identity authentication, authorization, streams and applications (see figure 2).

- **Identity Authentication**: confirms the identity via existing user accounts on social network providers. In this stage, the third party application can use the authentication services provided by social networks side (e.g., OAuth[3]) to authenticate users. It means that users are not required to create their profile on third party application, but using their existing profile on a specific social network to login. In this convenient way, the third party application can increase their number of users. After the authentication, the third party application can retrieve basic profile information (e.g., name, age, gender, friend list, etc.) about the user from a social network.

- **Authorization**: manages the access right to social networks' data. Except the basic profile information obtained from the identity authentication stage, normally a third party application wants to access other user's data on a social network, such as photos, messages, posts and so on. In this stage, the application lauches a request to access the data that it wants, and due to the agreement of the user the application now can access different user data in a specific social network.

- **Streams**: allows third-party application to send data to or obtain data from users' activity streams. This is considered as a channel to enable the communication between social networks and third-party sides.

- **Applications**: in the application level, the third-party side can develop its own social features and thus enrich the social networking sites

There are three types of data in user data. The first one - identity data identifies the user in social network, containing identity, profile information and privacy policy. Social graph data describes the relationships in the online social community (e.g., family, friends, groups that the user participates in). At last, content data describes all data created through user activities, such as photos, videos, status, and messages.

For each social network, the implementation of these API can vary according to the protocol and technology the social network uses.

2.3.1 Facebook API

Facebook Platform (see figure 3) uses OAuth 2.0 for authentication and authorization. OAuth 2.0[4] is a simplified standard that let third-party application obtain authorization tokens from Facebook platform. First, on the side of third-party application, a user authenticates using Facebook as an identity provider. Next, Facebook issues a token that lets the third-party application access the user's basic profile information, including name, picture, gender, and friend list.

As mentioned before, the access right can be extended depends on the agreement of the user and the requirement of an application. At this time, the third-party application will

[3] http://oauth.net/
[4] http://oauth.net/2/

send a request to extend the access right based on its requirements, and the user must allow the application to access his or her data on the social network platform. For example, to read posts with location information, you need "user_photos" or "friends_photos" permissions[5].

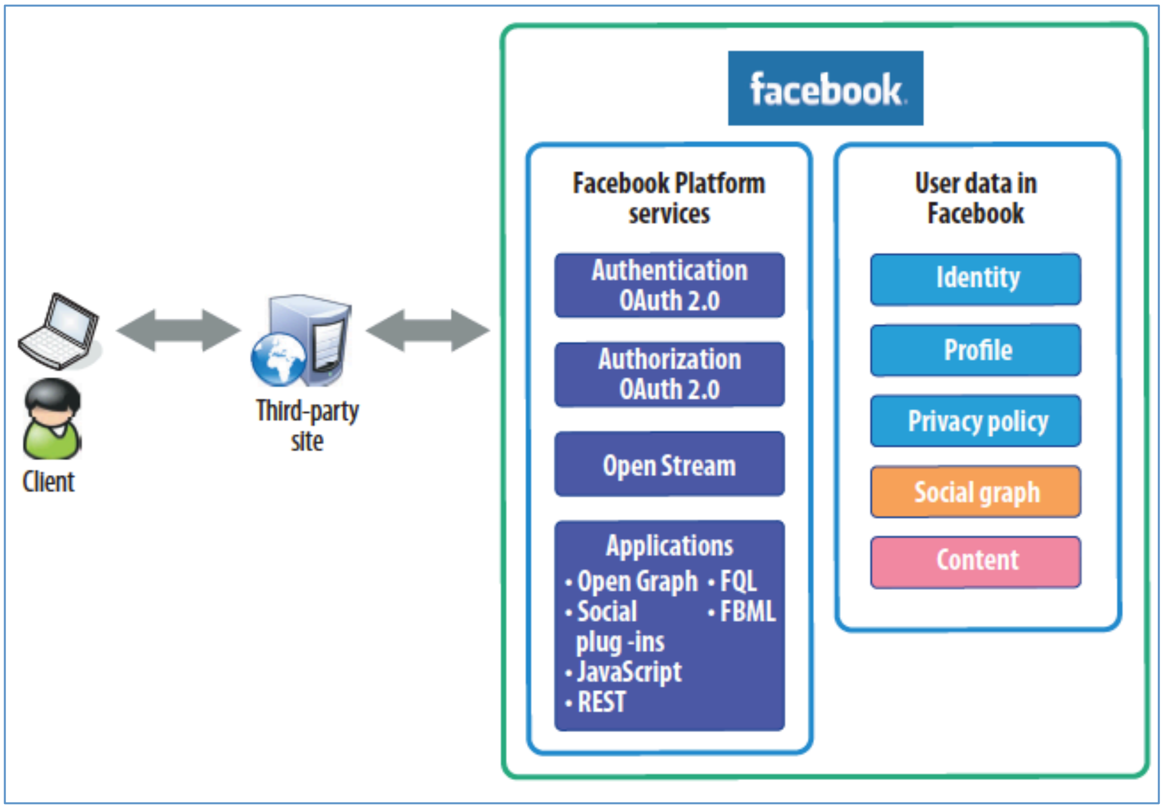


Figure 3 - Facebook Platform services

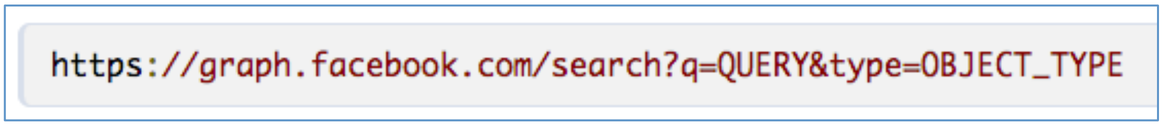
The Open Stream API lets third-party sites read and write to users' activity streams. This API supports multiple-stream publishing methods as well as the Atom feed standard[6].

Facebook provides a series of API to assist developers in creating social applications. The primary API is Open Graph, which lets third-party applications read and write content objects - photos, friends, and so on - and the connections among them in Facebook's social graph. Open Graph is used a lot in our thesis application, but FQL (Facebook Query Language) is also used to improve the searching in some cases. FQL provides a quick and easy mechanism to query Facebook user data without using API methods. In our application, we use both FQL and Graph Search API to obtain data from Facebook Platform. FQL query and Graph Search API both return data in JSON format.

- **Facebook Graph Search API[7]**

[5] https://developers.facebook.com/docs/reference/fql/location_post/
[6] https://developers.facebook.com/blog/post/225/

We can search over all public objects in the social graph with https://graph.facebook.com/search. The format is:



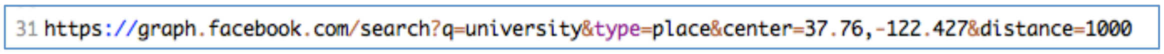
```
https://graph.facebook.com/search?q=QUERY&type=OBJECT_TYPE
```

Figure 4 - Facebook Graph Search Query

In the Graph Search API Query, we can search by type of object, such as people, pages, events, groups, places, check-ins, and object with location. The query contains 2 parameters

- q=QUERY. Here we can pass a keyword for searching
- type=OBJECT_TYPE. Here we can pass a type of object that we need to search, such as people, pages, groups, etc.

Moreover, we can narrow our search to a specific location and distance by adding the center parameter (with latitude and longitude) and an optional distance parameter, such as



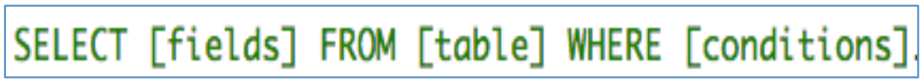
```
https://graph.facebook.com/search?q=university&type=place&center=37.76,-122.427&distance=1000
```

Figure 5 - Facebook Graph Search Query with Location

The above query will search any university in the location with a specific pair of latitude (37.76) and longitude (-122.427) and a given diameter (1000 meters).

- **<u>Facebook FQL API[8]</u>**

A FQL query is formalized as follow:



```
SELECT [fields] FROM [table] WHERE [conditions].
```

Figure 6 - Facebook FQL Query

The query looks like an SQL query, but the FROM clause can contain only a single table. The sub-queries can be inserted into the IN keyword that is located in SELECT or WHERE clauses. These sub-queries cannot point to the variables outside the query's boundary. Moreover, the query must be indexable field that is written very clear in the Facebook document. Operators, such as simple match or boolean (AND, OR, etc.), ORDER BY and LIMIT clauses are provided in Facebook FQL query, and thus make it powerful to process data.

[7] https://developers.facebook.com/docs/reference/api/search/
[8] https://developers.facebook.com/docs/technical-guides/fql/

Facebook Platform also supports the multi-query FQL, allowing gathering many FQL queries in one call and returns the data at one time. This enables us to process a more complex data in a faster way.

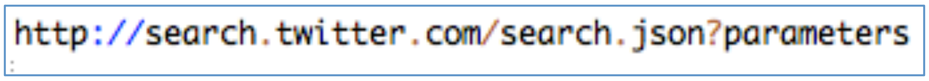
In our application, we use interchangeably Facebook Graph Search API and Facebook FQL API to implement our services.

2.3.2 Twitter API

The advantage of Twitter is that this platform attracts a large set of people updating news up-to-minute. The average number of tweets posted on Twitter a day is 200 millions tweets. In Twitter, streams of tweets can make up a lot of trends or topics that people follow. By using hashtags or mentions, we can search information in a quick and efficient way. For example, we can search what people are talking about VUB by using hashtag #VUB. Even more in details, we can search what people are talking about VUB with a positive thinking by adding another hashtag of feeling like #excited #enjoy.

The Twitter Platform offers a set of API[9]; each of them represents a facet of Twitter. In our application, we use the Search API in almost cases, because it can search tweets by using specific keywords, finding tweets of a specific user, or provide access to data around trends. We also use Streaming API to obtain the geo-tagged tweets from a certain region.

The request is formed as the follow GET Http request



```
http://search.twitter.com/search.json?parameters
```

Figure 7 - Twitter API Search Request

Parameter	Description
q (required)	A UTF-8, URL-encoded search query of 1,000 characters maximum, including operators. Queries may additionally be limited by complexity.
geocode	Returns tweets by users located within a given radius of the given latitude/longitude. The location is preferentially taking from the Geotagging API, but will fall back to their Twitter profile. The parameter value is specified by "latitude,longitude,radius", where radius units must be specified as either "mi" (miles) or "km" (kilometers). Note that you cannot use the near operator via the API to geocode arbitrary locations; however you can use this geocodeparameter to search near geocodes directly. A maximum of 1,000 distinct "sub-regions" will be considered when using the radius

9 https://twitter.com/search-home

	modifier.
lang	Restricts tweets to the given language
include_entities	When set to either true, t or 1, each tweet will include a node called "entities,". This node offers a variety of metadata about the tweet in a discrete structure, including: urls, media and hashtags.
until	Returns tweets generated before the given date.
rpp	The number of tweets to return per page

Figure 8 - Twitter Search API Parameter

2.3.3 Google Plus API

Google Plus Platform also uses OAuth 2.0 to allow third-party side access to its API. All applications that access a Google API must be registered. The result of this registration is a set of values (such as a client ID and client secret) that are known to both Google and the application.

Before your application can access private data using a Google API[10], it must obtain an access token that grants access to that API. As the Facebook does, a single access token can grant varying degrees of access to multiple APIs.

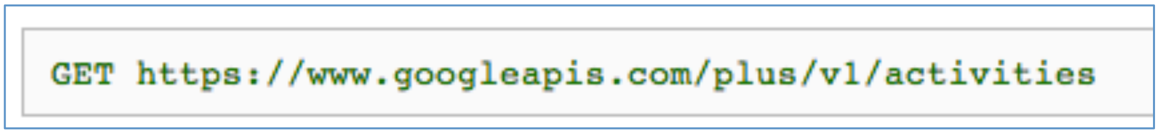
Some requests require an authentication step where the user logs in with their Google account. After logging in, the user is asked whether they are willing to grant the permissions that your application is requesting. If the user grants the permission, the Google Authorization Server sends your application an access token (or an Authorization code that your application can use to obtain an access token). If the user does not grant the permission, the server returns an error.

Access tokens are only valid for the set of operations, resources and in a limited time. If the application needs access to Google API beyond the lifetime of a single access token, it can obtain a refresh token. A refresh token allows the application to obtain new access token.

To search public activities, we use Google Search API[11]. The results are organized by paginating, but we cannot use a page token longer than 5 minutes. If the set of results is too large, we should restart pagination and re-send the request.

[10] https://developers.google.com/+/api/
[11] https://developers.google.com/+/api/latest/activities/search

The request is formed as the following GET Http Request.



```
GET https://www.googleapis.com/plus/v1/activities
```

Figure 9 - Google Plus Search API Request

The following parameters can be added at the end of the request to search a specific activity.

Parameter	**Description**
query (required)	Full-text Search query String
language	Restricts the search to the given language
lang	Restricts tweets to the given language
orderBy	Specifies how to order search results. The value could be • "best": Sort activities by relevance to the user, most relevant first. • "recent": Sort activities by published date, most recent first.
fields	Specifies which fields to include in a partial response.
maxResults	The maximum results to include in the response.

Figure 10 - The Google Plus Search API Parameter

2.4 SCOUT

SCOUT [22] is a framework developed to allow mobile applications to become aware of mobile user's context, such as personal information (profile, device characteristics, etc.), and current environment (people, objects, and locations in nearby place). The software framework developed in this thesis utilizes SCOUT to collect the mobile user's context.

SCOUT utilizes the Semantic Web Technologies to represent and retrieve context data. Below, we shortly summarize relevant Semantic Web concepts (RDF, SPARQL, etc.). Afterwards, we summarize the SCOUT framework itself.

2.4.1 Semantic Web

At the beginning of the Web history, the web, Web 1.0, could be considered as a static web. In other words, this kind of web only let us read or search information, and thus the interaction only occurred in one-way direction.

Web 2.0 provides us with more interactivity. Now, users could collaborate or interact with each other in online communities, and add their own contributions to update the website contents. In this period, there are many emerging terms, such as blogs, social networks, RSS, wikis, bookmark, mash-ups, and so on.

The explosive growth of Web 2.0 makes the web resources become a mess and chaos with tons of unorganized information. The biggest challenge that the web faces today is that information is structured to be read by humans, and not for machine interpretation. For example, typical search engines are based on keywords provided by humans to find relevant information, and do not understand what the particular search task is about.

The next evolution, web 3.0 (or Semantic Web) addresses this problem by adding semantics or meaning to the data. In doing so, data becomes readable and processable by machines. As the result, the Semantic Web will enable computers to autonomously implement tasks and find answers that currently require users' involvement.

The Semantic Web is a web of data instead of document like the current web. The Semantic Web provides a common framework that allows data to be shared and reused across applications. In this framework, shared ontologies are used to establish meaning (semantic) of the object and meaning of relations between objects. Understanding data in that way, machine can solve problems using many different kinds of information (e.g., instead of searching data about restaurant, Semantic Web can provide more relevant information, such as menus, cuisine styles, chefs, wine list) or make reason about it (e.g., deduce new facts based on the existing data).

The Semantic Web provides a set of technologies for representing, storing, and querying information. The next sections will describe these technologies in more detail.

2.4.2 RDF, RDFS, OWL, SPARQL

- **RDF**: The Resource Description Framework is a framework to describe resources in the World Wide Web. A resource can be a physical such as a building in real life or a concept, virtual entity like specific roles (e.g., the Belgium King). An Internet resource is defined by a unique URI (Uniform Resource Identifier). RDF describes those resources in a RDF statement, which contains subjects, predicate, and object. Subjects and predicates can be resources and is represented by URI references. Object can be a resource, or can be a literal (e.g., a string or a number). RDF is not designed to display for human consumption; it is readable and processable by machine. Written in XML format, RDF can be used to exchange data between different platforms and applications.

- **RDFS**: RDFS is an extension of RDF, in which a new notion of a class that is a type of thing is introduced. For example, a cat and a dog are members of the class Animal. The purpose of this notion is to indicate what kind of thing a resource is. Other notions are also introduced in RDFS, like domain and range, allowing applications to make inferences from statements about the type of things, and providing vocabularies.

- **OWL**: Web Ontology Language is, like RDFS, a language to define vocabularies. OWL is built on top of RDF. It enables greater machine interpretability by providing more vocabulary, thus is more expressive than RDFS. OWL adds, among others, cardinality, relations between classes and more properties.

- **SPARQL**: is a query language for RDF and have a similar syntax as SQL language. We can make a SPARQL query to diverse data sources on the web. The conjunction and disjunction features of SPARQL enable making query of required and optional graph patterns. Moreover, SPARQL is equipped with constraints and functions to create more complex queries. SPARQL queries can return in results sets or RDF graphs.

2.4.3 SCOUT Framework

SCOUT is a mobile application framework that supports linking physical entities to online semantic data sources. Our application utilizes SCOUT framework, running on a mobile Android device. By exploiting the rich personal information these device capture (for example, user preferences, or social networks data), we can personalize the provided content and functionality based on both personal profile and the user environment, provided by SCOUT framework.

Understanding SCOUT is a must and a very first step to do in developing our application. SCOUT detects entities in the user's surroundings using two methods. The first one is detection and sensing technologies, such as Quick Response (QR) codes or RFID, to let mobile devices detect tagged physical entities (people, places, and things) in a short range. In a mean while, online services such as LinkedGeoData (http://linkedgeodata.org) can be used to obtain entities near the user's current GPS location.

One of the most interesting aspects is that SCOUT doesn't store and manage data in a centralized server; instead, it uses web itself as an information system to obtain useful environmental information. Often, information on the physical entities is already available on the web - for instance, on webpages or in Semantic Web sources. Consequently, SCOUT accesses the online information, rather than storing it in its own server.

The SCOUT framework consists of several distinct layers. Each one is responsible to work independently for its own purpose, and is described in the below sections.

2.4.3.1 Detection Layer

The detection layer is responsible for detecting physical entities in the user's surroundings and extracting references to those entities' online semantic description (for example, an RDF description).

2.4.3.2 Location-Management Layer

The location-management layer interprets the raw information received from the detection layer. It determines whether detected entities are nearby the user or other detected entities, and when they're no longer nearby.

The location-management layer notifies the environment layer of nearness and remoteness events, along with the employed criterion, the entities' (approximate) locations, and references to their online data sources.

2.4.3.3 Environment Layer

The environment layer allows applications to make queries related to the user and all physical entities in his or her vicinity, which is managed by the environment model. The environment model contains the user and the proximity model.

The user model provides the user's personal information, such as preference, characteristics, and device information. Moreover, information from other applications can also be useful, like a personal agenda, schedule or calendar. This information is expressed by using ontologies such as CCPP[12] and FOAF[13] (Friend Of A Friend).

[12] http://www.w3.org/Mobile/CCPP/
[13] http://xmlns.com/foaf/spec/

The proximity model encodes positional information about the user's environment. It keeps time-stamped positional relations between the user and physical entities, together with references to those entities' associated data sources. A positional relation represents the fact that an entity is, or has been, nearby the user or another entity.

The environment model encompasses the user and proximity model, and extends them with information obtained from the physical entities' online semantic sources. Applications query the environment model using the query service. By using Semantic Web technology, it allows integrating information from different heterogeneous data sources by relying on the re-use of well-known ontologies and a unique resource identity via URIs.

3 Related Work

This chapter describes approaches related to the generic software framework proposed in this thesis. We describe relevant related work in three sections: full integration of social media and physical world in bi-directional way, extracting meaningful data from social media, and integrating multiple social networks.

3.1 Integration of Social Media and Physical World

The integration of social media data and physical world data is possible in two ways. The first one is from Social Media to Physical World. By being aware of the user's context, a computing system can make deduction about the current situation of the user, and thus can give the user recommendations and support based on relevant data exploiting from social media sources. In an opposite way (from the Physical World to Social Media), the user's context data could be collected and shared on social medias, and thus enriches the online community. The design of these applications was based on context-awareness.

However, almost recent researches only focus one of these two directions. Our thesis consists of both above directions to provide a full integration of social media and physical world data.

3.1.1 From Social Media to Physical World

This direction is to find relevant context data from social media to support the user in his or her context. There are many researches focusing on this approach to leverage different scenarios.

WhozThat [2] uses phone proximity (e.g., via Bluetooth or Wi-Fi) to exchange social network (e.g., Facebook) IDs, which are then used to fetch personal profile from the social network. By doing this way, WhozThat allows to identify the people in the user's surroundings.

MobileClique [19] exploits the user's social network profile (e.g., Facebook) to bootstrap a mobile ad hoc network that includes nearby users' mobile devices. This ad hoc network allows users to communicate and exchange content (e.g., messages, media data, etc.) using a store-carry-forward technique.

SocialFlicks [3] is an application that displays recommended movie trailers that match the movie preference of one or more users jointly watching a common display. SocialFlicks consists of Stationary and Mobile Components. Mobile Component shares a user's Facebook ID with the Stationary Component using Bluetooth. The Stationary Component detects the presence of users and uses the Netflix[14] REST API Web service to

[14] http://en.wikipedia.org/wiki/Netflix

construct a common playlist of recommended movie trailers based on their Facebook profiles.

Context DJ [18] is another context aware music player application that adapts the music being played at a given location according to the identities of people present at the location's context. Context DJ pulls data from user profile located on the LAST.FM - a musical social network and then uses the LAST.FM algorithm to calculate the artists that most users have in common.

SOMAR [23] integrates Facebook social network mobile data and sensor data to propose activities to the user (e.g., concert or computer science seminar) based on the user's current location and preference. This approach is similar to our, but our framework can exploit more social network data (groups, photos, people, etc.) rather than only activities.

In our work, the context-relevant information is automatically presented based on the user's location and places in his vicinity; while the above researches give recommendations based on the user's profile (e.g., movie preference, personal information). Moreover, the above researches only focus on one-way integration, from social media to the physical world.

3.1.2 From Physical World to Social Media

This direction is to share user's context information (or information deducted from the context) to enrich the social media platforms. The sharing can happen automatically, and thus enable user to update information on social media easily. There are many researches focusing on this direction to leverage different scenarios.

IYOUIT [11] is a research focusing on logging life events. IYOUIT automatically collects context data based on user's routines, and facilitates an instant sharing of personal daily life activities within online communities (e.g., with Flickr for sharing photos, with Twitter for sharing status).

SOLE [25] is a similar approach that allows user to share living experience in mobile environments. With SOLE, users can share their experience of their daily activities.

CenceMe [17] is able to collect, classify and infer user's present status and activity from the mobile device's sensors and export this information, in real-time, into social networks (e.g., Facebook and MySpace). CenceMe uses Social Classifiers on central servers to correlate the user's current activities, visited places and establish new social links (e.g., identifying CenceMe buddies in a user's neighborhood).

[8] obtains a set of rich user context by using sensors available in a smartphone as well as inexpensive and small sensors externally connected via Bluetooth and embedded in clothes (e.g., sound recorder, light sensor, position accelerometer, etc.). Here authors, use the supervised learning techniques to determine user context. From a set of training

examples it is possible to induce a decision tree in order to classify contexts according to sensor readings. The identified context information then will be shared on well-known social networks (e.g., Twitter and Hi5).

ContextWatcher [14] is another application that enables mobile phone users to automatically record, store and use context information, e.g., for personalization purposes, as input parameter to information services, or to share with online community, or just to store them for future use or to perform statistics on your own life.

ZoneTag [1] is a mobile photo upload tool that provides tag suggestions based on personal history, and user's current context (e.g., geographic location and time). ZoneTag uses a location translation mechanism and a suggested tag system to automatically annotate the photo with the user's location data and then share online (e.g., on Flickr).

FlickrTag [21] is another photo tag recommendation tool. The difference with ZoneTag is FlickrTag not only bases on the user's context (e.g., personal history, location and time), but also uses the analysis of tag characteristics (e.g., trends, tag categories) stored in Flickr to support the users to annotate.

[15] proposes a solution to automatically tag the nearby people appearing on a same picture. In this proposal, the author uses a mapping between the mobile phones' unique identifiers and their online social profile to add tags automatically on the picture when it is posted on social networks (e.g., Facebook).

Micro-Blog [10] is able to generate and share geo-tagged multimedia (e.g., micro blog). This data can be browsed or queried through either an Internet map service (e.g., Google Map) or in physical space as a user moves through a location.

In our approach, we also support the user to share context information but in both way of automatic posting and semi-automatic posting. Automatic posting is implemented in some default setting, and the semi-automatic way allows user to decide whether these information could be posted or not.

3.2 Extracting Data from Social Networks

Extracting meaningful data from social networks is a complex process because of the myriad of raw facts and unstructured data or incomplete data expression. There are many approaches to deal with this problem.

Event Identification in Social Media [4] proposes an approach to identify events and their associated user-contributed documents over social media sites (e.g., Twitter). The author exploits the rich context associated with social media content, including annotations (e.g., title, tags), generated information (creation time), and defines similarity metrics for these context features. Afterwards, similar documents are grouped into clusters via a weighted ensemble-clustering algorithm by combining any these features. Each cluster corresponds to one event and its associated social media documents.

Fujisaka [9] proposes a method for the detection of unusual crowding in physical locations from existing blog community. Via the analysis of common patterns of occurrence in each region over a specified time period employing K-means based micro-blog clustering, this method achieves the extraction of useful and interesting movement patterns, reflecting the occurrence of critical events in a geographic region. Experimental evaluation of the proposed method uses a real dataset collected from Twitter.

Many researches related to extract geo-tagged photos from social networks (e.g, Flickr, Panorama) to deduce the meaningful information. Photo2Trip [16] is able to generate travel routes from geo-tagged photos on Panorama for trip planning. R.Ji et al. [12] report a work on mining famous city landmarks from blog for personalized tourist suggestion. Q.Zhao et al. [24] propose detecting and framing events from the real world by exploiting the tags supplied by users in Flickr photos.

These proposals focus on extracting information from social networks and digesting them to produce knowledge (e.g., discover unusual crowding, or detect popular events). We focus on finding the social media data that is directly related to the user context (e.g., finding events, groups that take place in the user surroundings and the user's friends participate in). In our framework, we use the context matching function to determine the context-relevance of social media data. The context matching function compares pairs of properties in both the current user context and the social media data.

3.3 Integrating Multiple Social Networks

So far, there is no research focusing on integrating multiple social networks. Although there are many commercial mobile applications, allowing integrating multiple social media platforms (e.g., oneall[15], sharekit[16], etc.), these works do not create a general framework capable to support any social network. More specifically, each of the data types provided by the supported social media platforms are simply retained and put in separates places of the interface. One of the biggest problem is that different social networks provide different data types, structures, concepts, and services. In our approach, to deal with this problem, we build a shared vocabulary among social networks in order to create a unified data representation. After relevant-context data is extracted from social media platforms, the data will be converted into one of these shared common data type.

[15] http://www.oneall.com/
[16] http://getsharekit.com/

4 Approach

This chapter elaborates on the generic and mobile framework presented in this thesis, which supports plugging in any social media platforms. First, we present the framework architecture. We discuss how to create a set of shared common data type to resolve the conflicts or overlaps between different data types of multiple social media networks. To showcase the generality of the framework, we supply support for Facebook, Twitter and Google Plus, although other social media platforms can be plugged in as well. We also investigate methods to extract meaningful data related to user's context by formulizing requests to access resources of supported social media platforms. Afterwards, we will show how to process the social network data in order to find the most relevant-context data. Finally, we discuss how the User Interface is built based on the common data type structures.

4.1 Architecture Overview

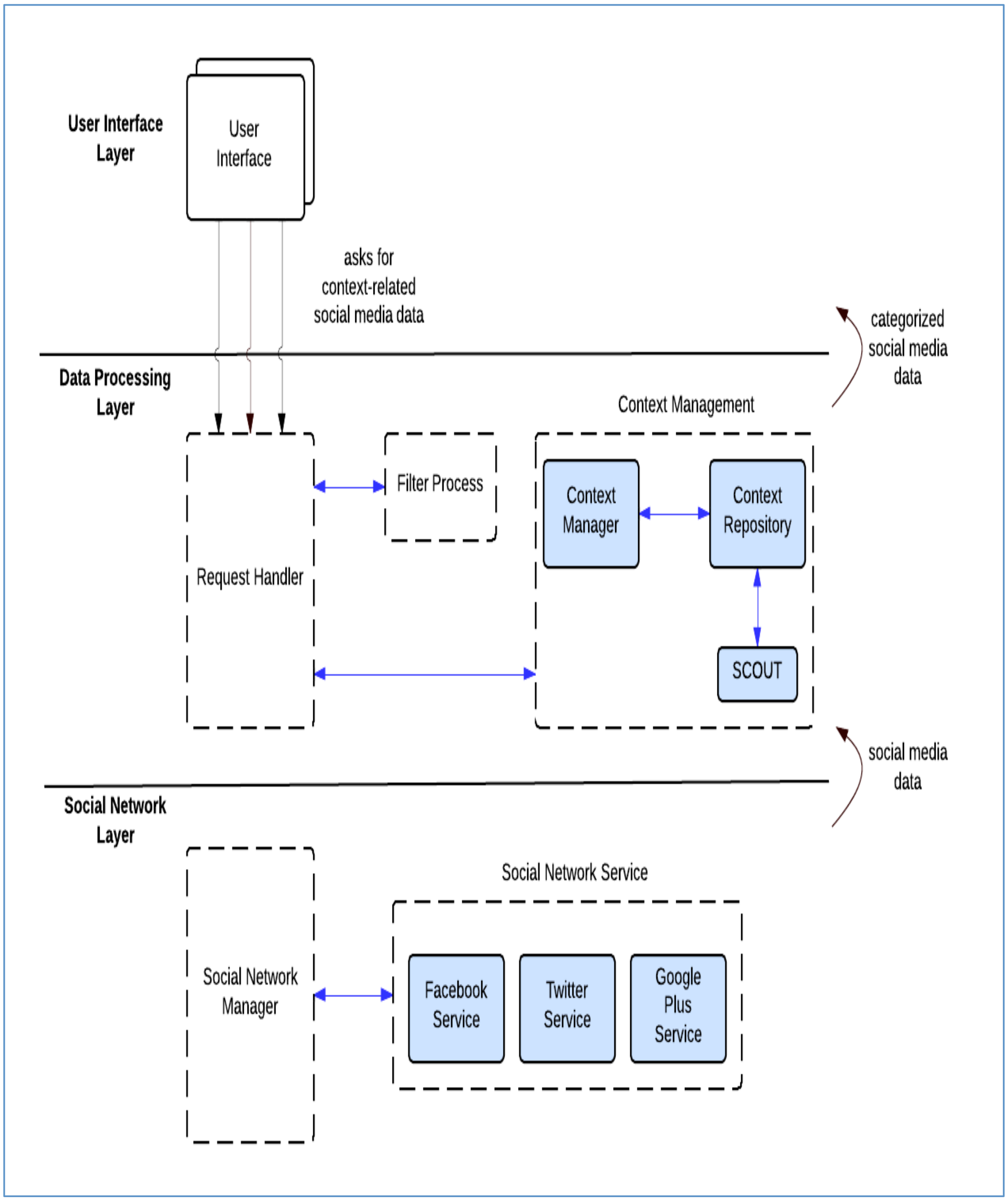


Figure 11 - Architecture Overview

This section explains and describes the architecture of the developed framework (illustrated in Figure 11). The architecture consists of several distinct layers, separating the different concerns. In doing so, the user interface is decoupled from the application logic, and each layer can contain interchangeable components. There are 3 layers: Social Network Layer, Data Processing Layer, and User Interface Layer.

The bottom layer, the **Social Network Layer**, is responsible to find the relevant-context data from various social network platforms. The context-awareness is applied when formulating the request to the social media platforms. This layer compromises two sub components: the Social Network Manager and the Social Network Service. The goal of **Social Network Manager** is to manage all the online social accounts. The **Social Network Service** provides a shared set of methods to access the resources of a specific social network platform (e.g., Facebook, Twitter, and Google Plus). These methods could be requesting for the user profile information, finding friends who has visited a specific place and so on (see section 4.2). Different platforms return different data type structures, and thus could lead to the conflicts between them. We resolve this problem by constructing a set of shared common data type (see section 4.2.1). In this layer, the data obtained from social network platforms is converted to one of these common data types, and then the converted data is sent up to the next layer - Data Processing Layer.

The goal of **Data Processing Layer** is to provide the user's context information (e.g., the nearby places, buildings) and categorize the Social Media data, obtained from the Social Network Layer. The **Request Handler** plays a role to manage and distribute the request (e.g., a request for finding relevant context data, a request for finding nearby buildings, etc.). This component ensures to prepare data before sending a request, handle when data is coming successfully, or gives an error when failing to receive data. The **Filter Process** is responsible to rank the social media data based on the user's context (e.g., the nearby entities information, the user's current location, and time), the online user profile preference (e.g., favorite movies, music, sports). The **Context Manager** provides and stores the user's context (e.g., the user's current location, all visited nearby entities within a day). The user's context information can come from any Context Acquisition system (e.g., SCOUT framework) that is integrated into the Context Repository - a generic context repository (see section 4.3.1). Besides, the Context Manager implements the auto-posting mechanism, as well as the semi-auto posting mechanism (see section 4.3.1).

The **User Interface Layer** enables the interaction between the user and our system. This layer utilizes the ranked social media data list to present the most relevant context information to the user.

Below, we shortly summarize the data flow between these layers and their inner components. The interactions are described by using the UML sequence diagram as follow:

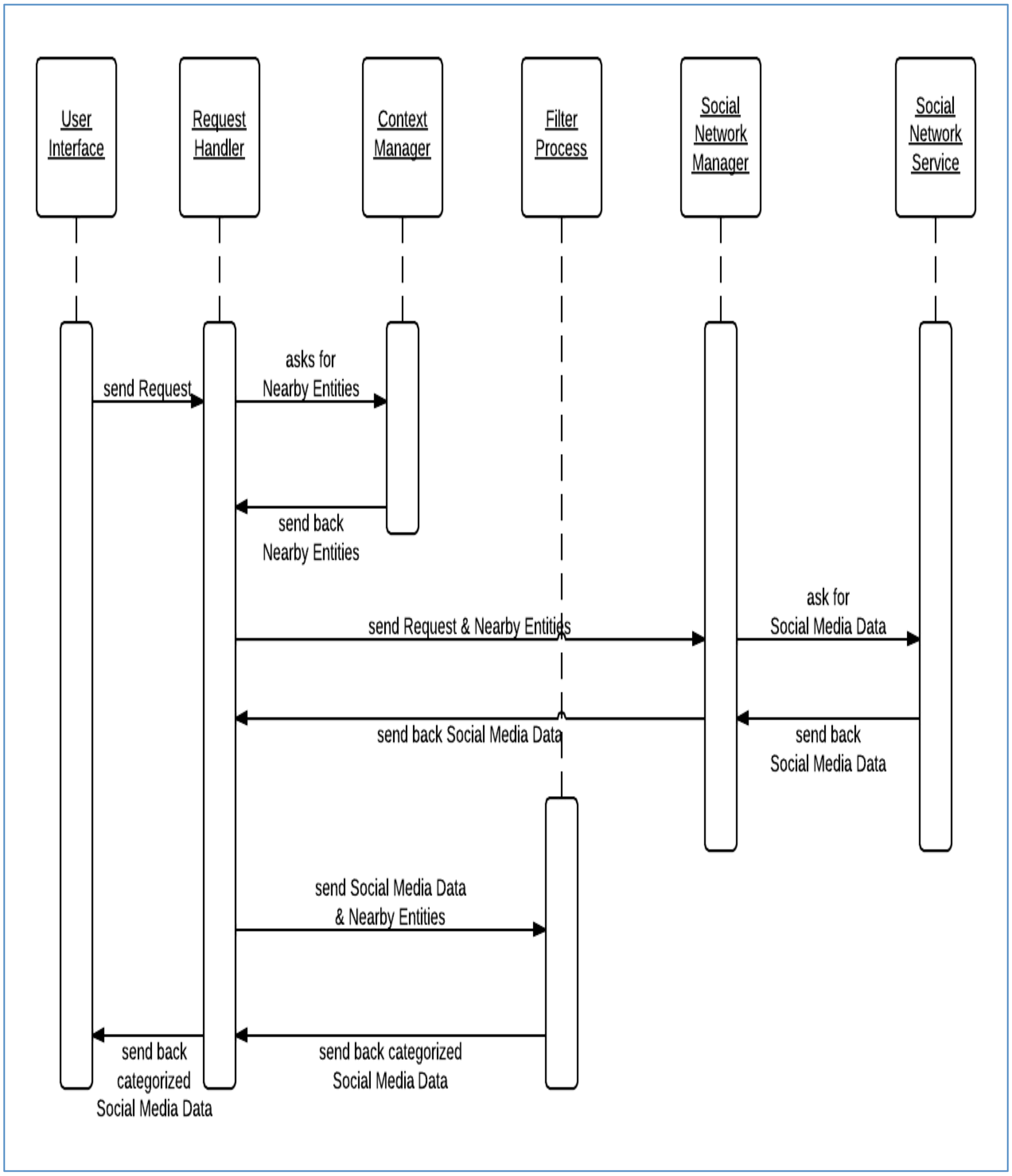

User Interface
Request Handler
Context Manager
Filter Process
Social Network Manager
Social Network Service
send Request
asks for Nearby Entities
send back Nearby Entities
send Request & Nearby Entities
ask for Social Media Data
send back Social Media Data
send back Social Media Data
send Social Media Data & Nearby Entities
send back categorized Social Media Data
send back categorized Social Media Data

Figure 12 - The data flow between layers

Given a specific user request for finding a relevant-context data type (e.g., check-ins data type - finding friends who visited a nearby place in the user's surroundings), the User Interface Layer sends the request to the layer Data Processing Layer.

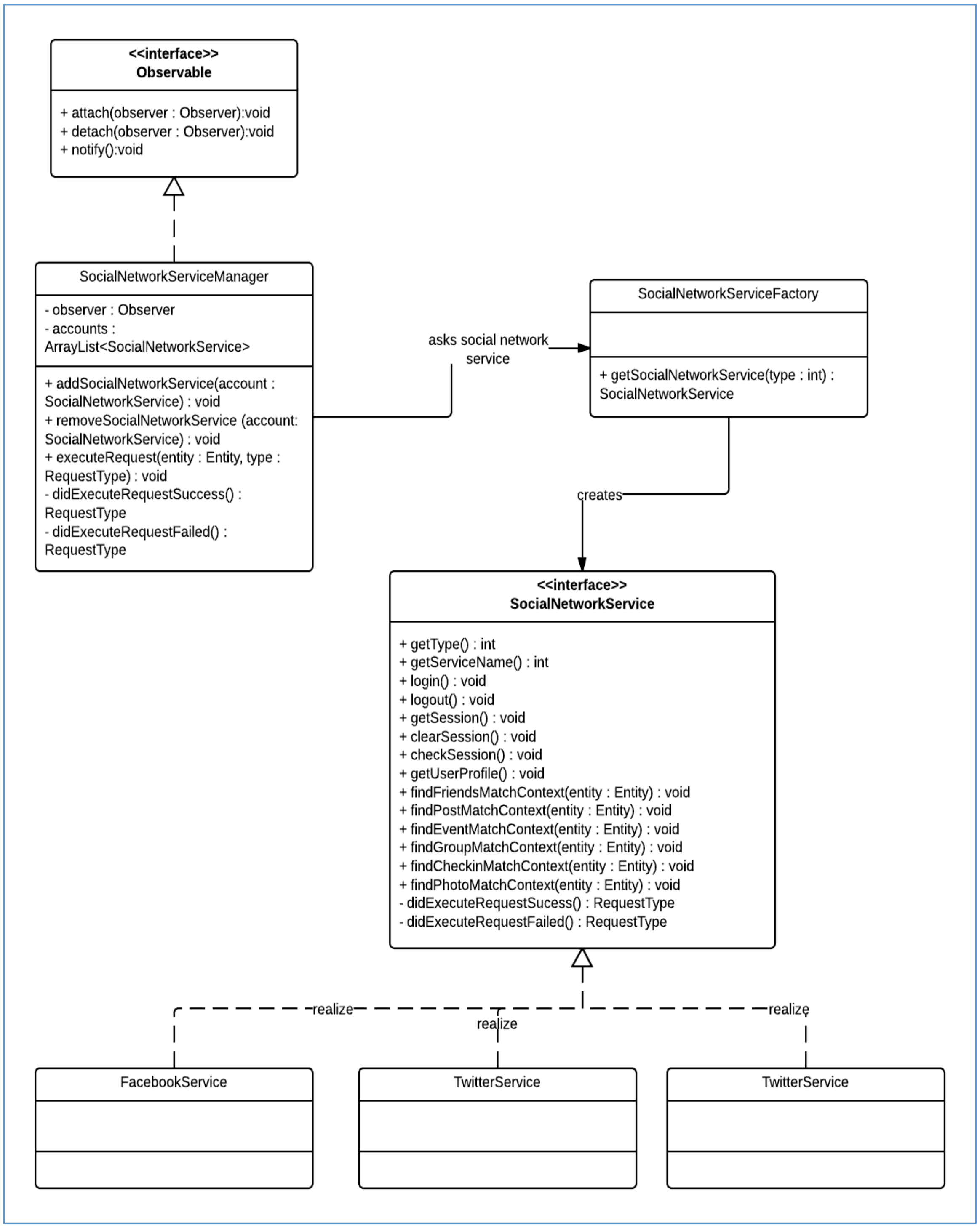
At the Data Processing Layer, the Request Handler receives the request, and before continuing sending this request to the next layer, it asks the Context Manager for the nearby entity information that the user is currently interested in. Afterwards, the Request

Handler sends the request along with the nearby entity information to the Social Network Manager. Based on the request and the nearby entity information (e.g., name, title, longitude, latitude, social page ID), the Social Network Manager will ask for the required resource from all social networks it has. Accessing the resource happens at the Social Network Service. After getting the relevant context social media data, the Social Network Manager keeps sending this data back to the Request Handler.

The Request Handler receives the social media data and sends this data along with the nearby entity that the user is currently interested in to the Filter Process. The Filter Process implements the ranking of the social media data, based on the online user profile information (e.g., user's preference hobbies and activities) and the user's context information (e.g., current location, time, nearby entity's name). The Request Handler receives the categorized social media data and keeps sending the data back to the User Interface Layer. This layer displays data based on the relevance degree. For example, the most relevant data will be on top, and then the less relevant data will come after. The none-relevant data is not displayed.

In the sections below, we detail each of these layers.

4.2 Social Network Layer



<<interface>>
Observable
+ attach(observer : Observer):void
+ detach(observer : Observer):void
+ notify():void
SocialNetworkServiceManager
- observer : Observer
- accounts : ArrayList<SocialNetworkService>
+ addSocialNetworkService(account : SocialNetworkService) : void
+ removeSocialNetworkService (account: SocialNetworkService) : void
+ executeRequest(entity : Entity, type : RequestType) : void
- didExecuteRequestSuccess() : RequestType
- didExecuteRequestFailed() : RequestType
asks social network service
SocialNetworkServiceFactory
+ getSocialNetworkService(type : int) : SocialNetworkService
creates
<<interface>>
SocialNetworkService
+ getType() : int
+ getServiceName() : int
+ login() : void
+ logout() : void
+ getSession() : void
+ clearSession() : void
+ checkSession() : void
+ getUserProfile() : void
+ findFriendsMatchContext(entity : Entity) : void
+ findPostMatchContext(entity : Entity) : void
+ findEventMatchContext(entity : Entity) : void
+ findGroupMatchContext(entity : Entity) : void
+ findCheckinMatchContext(entity : Entity) : void
+ findPhotoMatchContext(entity : Entity) : void
- didExecuteRequestSucess() : RequestType
- didExecuteRequestFailed() : RequestType
realize
realize
realize
FacebookService
TwitterService
TwitterService

Figure 13 - The Social Network Layer Core Classes

This section is accompanied by UML class diagram (figure 13) showing the core classes used in the implementation. The layer supports integration of multiple social networks into our framework. Given a request from the Data Processing Layer, this layer returns a set of relevant context data from all integrated social networks. There are two main parts in this layer.

The first one is SocialNetworkManager class, which manages the user's social media accounts (e.g., adding a new account, remove an existing account). When receiving a request from the top layer, for each account, the Social Network Manager is able to identify a social network accordingly, and make a call to the second part to implement the service request. To gain performance improvements when dealing with many requests at a time, each Social Network service request runs in its own thread. The observer pattern[17] is applied in this case in order to notify when the social media data is found. The SocialNetworkManager class implements the observable interface, and notifies all registered observer classes about the success or failure while searching social media data.

The second part, namely Social Network Service, is responsible to extract the social media data based on the user's context. To cope with multiple social network platforms, we utilize the Factory design pattern[18]. To support a specific social media platform, a Social Network Service component (e.g., Facebook Service, Twitter Service, etc.) is plugged in that allows contacting the particular platform via its API. By supplying a generic service interface - the SocialNetworkService interface, multiple platforms can be easily plugged in. The generic service interface provides a set of common methods to access the resources (e.g., photos, events, posts...) of various social network platforms. These methods includes:

Methods	**Description**
Login (Mandatory)	Allowing a user to login into a particular social network.
Logout (Mandatory)	Allowing a user to logout from a particular social network.
CheckSession (Mandatory)	Checking out a session is expired or not. A session is used to access to a resource on social network platforms, and it also determines how long the access right is permitted.
ClearSession (Mandatory)	Clearing a session. A session is used to access to a resource on social network platforms, and it also determines how long the access right is permitted.
GetUserProfile (Mandatory)	Giving the user profile information of a particular social network such as the user name, birthdate, gender, avatar picture, etc.

[17] http://en.wikipedia.org/wiki/Observer_pattern
[18] http://www.oodesign.com/factory-pattern.html

FindFriendsMatchContext (Optional)	Finding friends whose activities are related to the user's current location (e.g., living, working, studying nearby).
FindCheckInMatchContext (Optional)	Finding friends who visited places in the user's surroundings.
FindGroupMatchContext (Optional)	Finding groups with activities that are related to the user's location, such as the group belongs to a particular place, or organizations.
FindEventMatchContext (Optional)	Finding events that belongs to a specific entity, such as building, company, university that are located in the user's surroundings, or events that take place nearby user's current location.
FindPostMatchContext (Optional)	Finding all public posts that contains information about a specific entity, such as tourist attraction, organization, university that are located in the user's surroundings.
FindPhotoMatchContext (Optional)	Finding all photos and albums that friends have taken of places or things in the user's current surroundings.

Figure 14 - The set of common social network methods

In this list, there are some methods, related to the user's account, session and profile information that are mandatory to be implemented for all social network services. The remaining methods, related to finding the context information, are optional to be implemented. A specific social network service may not implement all above optional methods. This certainly is true, based on the fact that a resource may exist in a particular social network platform, but may not exit in other platforms. As a result, the method to access this resource may be applied to a particular social network service, but may be not applied to others.

When the developer wants to add a new social network into our system, he must do the following jobs:

1) Implement support for contacting the social network

He first has to register this social network in the Social Network Manager component. Secondly, a new Social Network Service subclass needs to be created to contact the new social network platform.

2) Potentially, introduce new common data type

Each social network has its own characteristics, and thus is possible to provide its own methods to access its resources (e.g., events, photos) accordingly. For example, in our system, we are now having Facebook and Twitter social networks, providing methods to find common relevant-context data (see the figure 7). Now, imagine we want to add support for LinkedIn, a social network about professionals and businesses. Clearly, the methods to access these topics (e.g., job offers based location[19]) are new with respect to our system. When a new social network emerges, the developer has to identify the provided methods to access its resources are already present in the list of existing methods or not. Therefore, there are two following cases:

- If the method already exists, he just has to implement this method for the new social network.

- If the method does not exist, besides implementing the new method for this new Social Network Service class, he has to look up the list to find out whether any existing social network is able to implement this method or not and if it's possible he has to write it.

3) Implement a conversion process

Although each integrated service thus supports a generic interface, platforms typically return different data type structures (e.g., posts, photos, groups, etc.). To deal with this, we constructed a set of common data types, representing the typical social media concepts found in social media platforms (see section 4.2). In this part, any data retrieved from a social network is converted to one of these common data types. The conversion process should be implemented in manually by the developer. The following steps could be an example of the conversion process, in which the developer receives a Tweet data type (a Twitter status update) and convert to the Post data type (our common data type represents a social network status update).

- Retrieving a Tweet data type from Twitter platforms in a JSON file format.
- Parsing the JSON file to extract the Tweet sub-properties (e.g., Tweet ID, Tweet text, etc.)
- Adapting accordingly these sub-properties into our common data type structure - Post data type.

The next session will explain how we construct a common data type in our system.

4.2.1 Data Type

One of the biggest challenges is reconcile various types of data found in existing social media platforms. Different social network provide different data types and concepts. For example, a term "friend" exists in Facebook, but in Twitter it turns out to be "following"

[19] http://developer.linkedin.com/documents/job-search-api

or "followers", in Google+ friendship refers to the people on circles. The difference is not only about the different names used by these social networks for the same concepts, but also the different meanings assigned to them. Friendships on Facebook are bidirectional, and thus require both parties to agree before connecting. On Google Plus, the relationship is express clearly by using circles. A circle in Google Plus stands for a relationship with a group of people. We group people in a circle depending on how we think of them in real life, such as family circle, close friends circle, etc. Now, you can share jokes with your close friends, photos with your family, and while your boss in the professional circle cannot see them. One person can be in several circles for example a cousin can be in both the family circle, and roommate circle. Twitter defines a relationship in even a different way. Twitter only requires unidirectional consent via the follow action. Twitter allows users to follow the other users' activities without any sort of approval, and Twitter does not divide the relationship clearly as Google Plus does. Therefore, it's not surprising that you can follow easily thousands people on Twitter in a very short time, but it's much more time consuming to get the same number of friends on Facebook or Google Plus.

Given this variety in social media data types, we need to construct common data types to reconcile overlaps or conflicts. In our system, we identify a property as a small unit that contains a name and a value (e.g., age or name properties). A resource consists of one or many properties and other sub-resources. A reconciled resource will be the combination of properties of similar resources from different social networks. Let's take two similar resources from two different social networks as an example, namely Facebook Post and Twitter Tweet. Both resources are about the status update that users use to share on Facebook and Twitter. A Facebook Post[20] has the following fields as follow.

Fields	**Description**
id	The post ID
from	Information about the user who posted the message
to	Profile mentioned or targeted in this post
message	The message
place	Location associated with a Post, if any
created_time	The time the post was initially published
message_tags	Objects tagged in the message (Users, Paged, etc.)
picture	If available, a link to the picture included with this post

Figure 15 - Facebook Post Fields

[20] https://developers.facebook.com/docs/reference/api/post/

A Twitter tweet[21] has the following fields as follow.

Fields	Description
id	The tweet ID
user	The user who posted this Tweet
current_user_retweet	The user's own retweet (if existent) of this Tweet
text	The status update
place	Indicate the place that the tweet is associated with
created_at	The time when this tweet was created
entities	Entities which have been parsed out of the text of the Tweet

Figure 16 - Twitter Tweet Fields

The two above tables show us that Facebook Posts and Twitter Tweets have many fields in common. We can take advantage of this to construct our own data type representing status updates. For example, the following figure (figure 17) illustrates how to reconcile tweets and Facebook posts.

Facebook Post	Twitter Tweet	Our System Post	Description
id	id	id	The id of a post
from	user	fromUser	The author of a post
to	current_user_retweet	toUser	The user who is targeted in this post
message	text	text	The status update content
place	place	place	The location associated with the post if any
created_time	created_at	created_time	The time when this post was created
message_tags	entities	objects	Objects that were tagged in this

[21] https://dev.twitter.com/docs/platform-objects/tweets

			post

Figure 17 - Constructing the shared Post data type

In this example, we create a sample data called a Post data type for both Facebook Post and Twitter Tweet. Now, we have a uniform data type to express the status update of both Facebook Post and Twitter Tweet. Our Post data type also contains every shared field potentially named differently. Naming these fields is up to the developer. In the scope of this thesis, we don't have a mechanism to convert different data structures into our own data structure. The developer implements the conversion and naming process in a manual way, based on the meaning and concept of these data types in different social networks. This conversion involves parsing the data, and converting it to our common data type.

In reality, reconciling different social media concepts costs a lot of time and deep thinking about their meaning, because a resource can contain multi-level sub resource, showing complicated sub-structures. When adding a new social network to our system, we should consider the existing data types and think if it is necessary to create a new data type or use the existing ones. Of course, each social network often has its own meaning, and serves a specific purpose, like Foursquare provides data type of places, coupons, promotion, and LinkedIn provides data type related to jobs, career information. Consequently, specialized data types of these social networks will probably not overlap, but basic data type like posts, status, and personal profile data are identical.

Aside from (potentially) introducing new common data types, the developer also needs to implement support for contacting the particular social network.

4.3 Data Processing Layer

The Data Processing Layer is responsible to provide the user's context (e.g., nearby entities, user's current location) and categorize the social media data based on the context relevance. There are three main components in this layer: Context Repository, Request Handler, and Filter Process.

The Context Repository provides the user context. This component allows integrating different context frameworks, e.g., SCOUT (see section 4.3.1).

The Filter Process is deployed to categorize the social media data from the most relevant context data to the least relevant context data (see section 4.3.3). The degree of relevance will be used in the User Interface Layer.

The Request Handler is considered as the entrance to obtain the requests from and return the results to other layers. For more detail of how the request is handled, please see the section 4.3.2.

The following figure shows the UML class diagram showing the core classes.

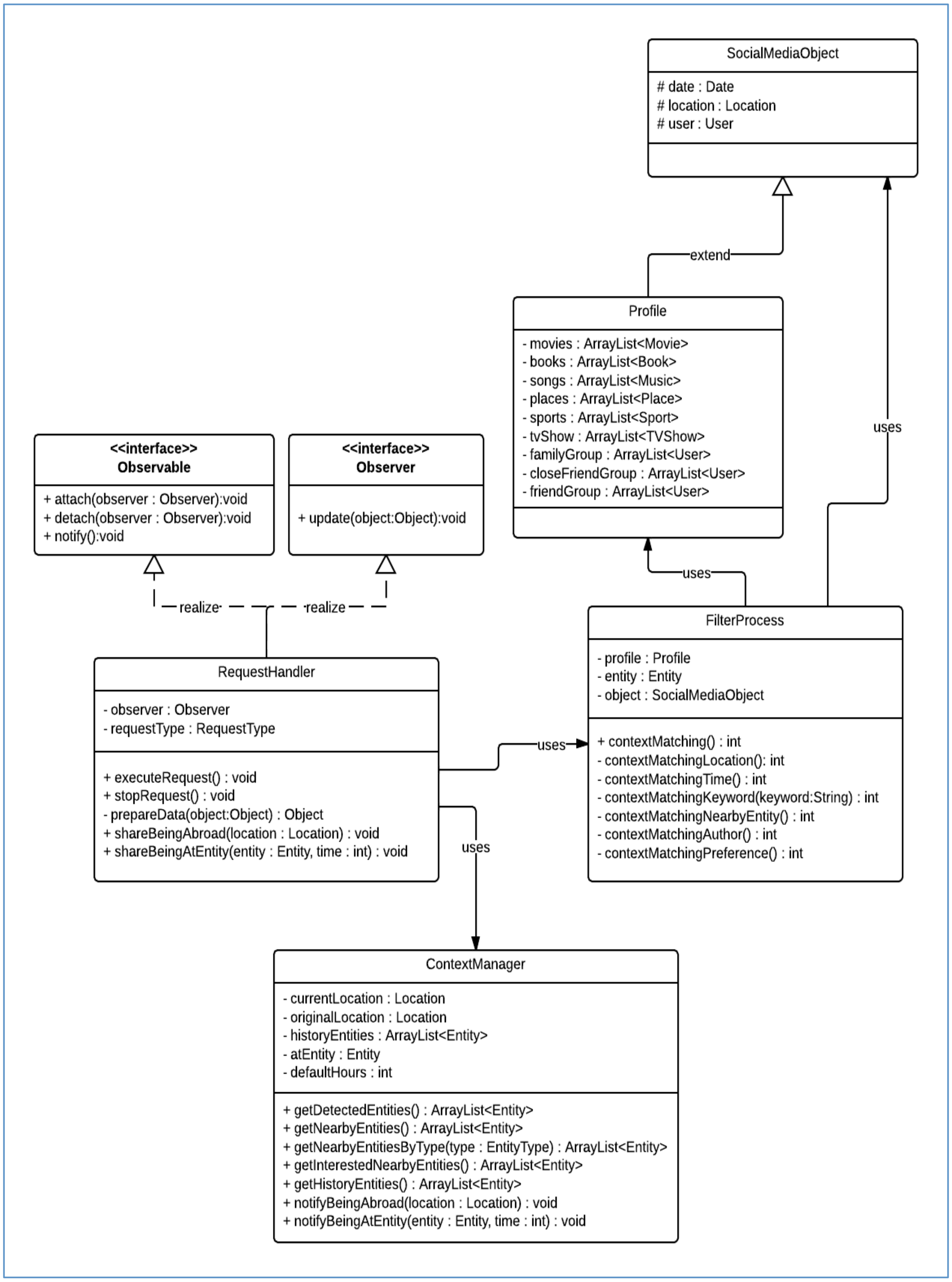

SocialMediaObject
# date : Date
# location : Location
# user : User
extend
uses
Profile
- movies : ArrayList<Movie>
- books : ArrayList<Book>
- songs : ArrayList<Music>
- places : ArrayList<Place>
- sports : ArrayList<Sport>
- tvShow : ArrayList<TVShow>
- familyGroup : ArrayList<User>
- closeFriendGroup : ArrayList<User>
- friendGroup : ArrayList<User>
<<interface>>
Observable
+ attach(observer : Observer):void
+ detach(observer : Observer):void
+ notify():void
<<interface>>
Observer
+ update(object:Object):void
realize
realize
uses
FilterProcess
- profile : Profile
- entity : Entity
- object : SocialMediaObject
+ contextMatching() : int
- contextMatchingLocation(): int
- contextMatchingTime() : int
- contextMatchingKeyword(keyword:String) : int
- contextMatchingNearbyEntity() : int
- contextMatchingAuthor() : int
- contextMatchingPreference() : int
RequestHandler
- observer : Observer
- requestType : RequestType
+ executeRequest() : void
+ stopRequest() : void
- prepareData(object:Object) : Object
+ shareBeingAbroad(location : Location) : void
+ shareBeingAtEntity(entity : Entity, time : int) : void
uses
uses
ContextManager
- currentLocation : Location
- originalLocation : Location
- historyEntities : ArrayList<Entity>
- atEntity : Entity
- defaultHours : int
+ getDetectedEntities() : ArrayList<Entity>
+ getNearbyEntities() : ArrayList<Entity>
+ getNearbyEntitiesByType(type : EntityType) : ArrayList<Entity>
+ getInterestedNearbyEntities() : ArrayList<Entity>
+ getHistoryEntities() : ArrayList<Entity>
+ notifyBeingAbroad(location : Location) : void
+ notifyBeingAtEntity(entity : Entity, time : int) : void

Figure 18 - The Data Processing Layer Core Classes

4.3.1 Request Handler

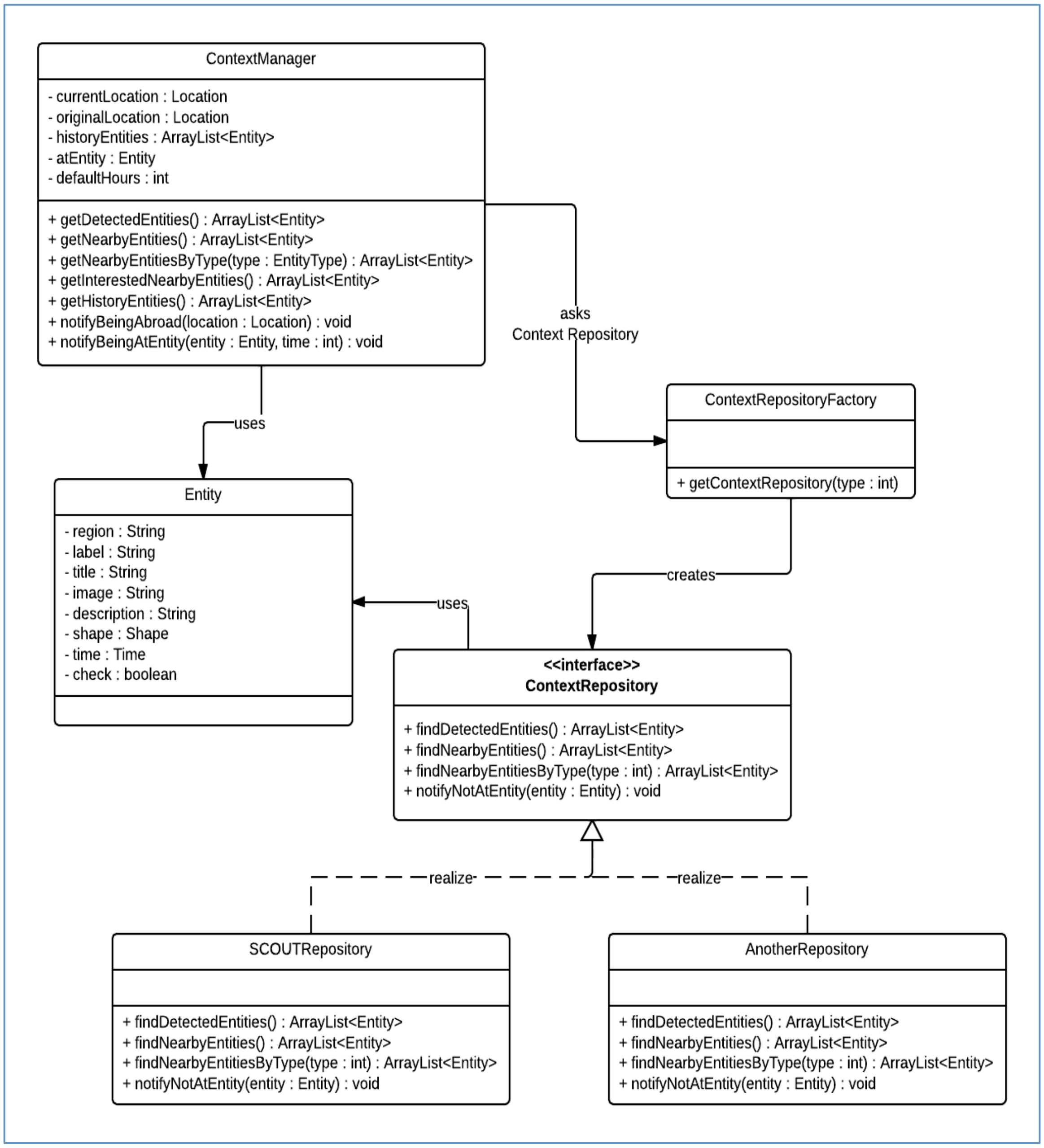
The figure 10 shows that we utilize the Observer design pattern for the RequestHandler class. In this case, the RequestHandler plays both roles the Observer and the Observable object. On the one hand, the RequestHandler class is an Observable when The User Interface Layer objects send a request to and register in order to get notified when the social media data is ready to display. On the other hand, the RequestHandler plays as the Observer role, and register with the SocialNetworkManager class in order to get notified when the social media data is found from any social network platform. The RequestHandler also get notified when the user is abroad and the amount of time that the user is at a specific place is over the predetermined default hour, whereby the user's location is shared online. The auto-posting mechanism is implemented by this way.

Moreover, each request is executed in a separated thread. Executing requests in the background ensures the user interface responsive. When a request is completed, the thread will wake up the Request Handler component, and then the Request Handler will send all relevant data to the Filter Process to find out the most relevant data before sending them back to the User Interface Layer. The Request Handler manages all requests into threads, for example it can spawn a new thread to implement a request, or it can interrupt and destroy all running threads. In doing so, the system is ensured to not be memory leak when the number of requests is increasing or the thread to implement a request is still running but we no longer need it.

4.3.2 Context Repository

The ContextManager class provides access to the mobile user's context. Besides, this class keeps the user's current location, the user's hometown location, and all visited entities during a day.

Below is the UML class diagram showing the core classes.



ContextManager
- currentLocation : Location
- originalLocation : Location
- historyEntities : ArrayList<Entity>
- atEntity : Entity
- defaultHours : int
+ getDetectedEntities() : ArrayList<Entity>
+ getNearbyEntities() : ArrayList<Entity>
+ getNearbyEntitiesByType(type : EntityType) : ArrayList<Entity>
+ getInterestedNearbyEntities() : ArrayList<Entity>
+ getHistoryEntities() : ArrayList<Entity>
+ notifyBeingAbroad(location : Location) : void
+ notifyBeingAtEntity(entity : Entity, time : int) : void
asks
Context Repository
uses
ContextRepositoryFactory
+ getContextRepository(type : int)
Entity
- region : String
- label : String
- title : String
- image : String
- description : String
- shape : Shape
- time : Time
- check : boolean
creates
uses
<<interface>>
ContextRepository
+ findDetectedEntities() : ArrayList<Entity>
+ findNearbyEntities() : ArrayList<Entity>
+ findNearbyEntitiesByType(type : int) : ArrayList<Entity>
+ notifyNotAtEntity(entity : Entity) : void
realize
realize
SCOUTRepository
+ findDetectedEntities() : ArrayList<Entity>
+ findNearbyEntities() : ArrayList<Entity>
+ findNearbyEntitiesByType(type : int) : ArrayList<Entity>
+ notifyNotAtEntity(entity : Entity) : void
AnotherRepository
+ findDetectedEntities() : ArrayList<Entity>
+ findNearbyEntities() : ArrayList<Entity>
+ findNearbyEntitiesByType(type : int) : ArrayList<Entity>
+ notifyNotAtEntity(entity : Entity) : void

Figure 19 - The Context Repository diagram

This component supports a generic context interface that can utilize different context acquisition systems (e.g., SCOUT) behind the scenes. Once again, the Factory design pattern is used in this case. By doing so, different context acquisition system can be plugged in without having to change other components in this layer (e.g., Filter Process). This way can ensure a separation of concerns among internal components. In the scope of this thesis, in the term of the mobile user's context, we focus on discovering physical entities (e.g., places, buildings) in the user's surroundings. The generic context interface contains the following methods to access the mobile user's context:

- **Detecting Entities:** detecting all entities in the user surroundings.

- **Nearby Entities:** finding out all nearby entities in comparison with the user location.

- **Search Entities by type:** finding out a specific kind of entities, such as restaurant, coffee shop, or university.

In practice, a wrapper class is created for each context acquisition system, which implements the specified interface - the ContextRepository interface and translates these methods to operations on the context acquisition system. For example, for SCOUT, this wrapper class translates the method invocations to queries to be sent to the SCOUT Query Service.

After retrieving the raw data of user's context from context acquisition frameworks, the Context Repository component will abstract these data in the entity data type. The entity data type is defined as a set of properties as describing in the following figure.

Property	Description
title	The name of an entity
label	Another name of an entity
type	The type of an entity, such as restaurant, coffee shop, sports accommodation, university, etc.
description	Detail information to describe the function of an entity
homepage	A link to a website of an entity
image	An image to describe the appearance of an entity
location	A pair of the longitude and latitude where the entity is located
time	The time that the entity is open or close
parent	A parent entity, identifying that an entity belongs to another entity.

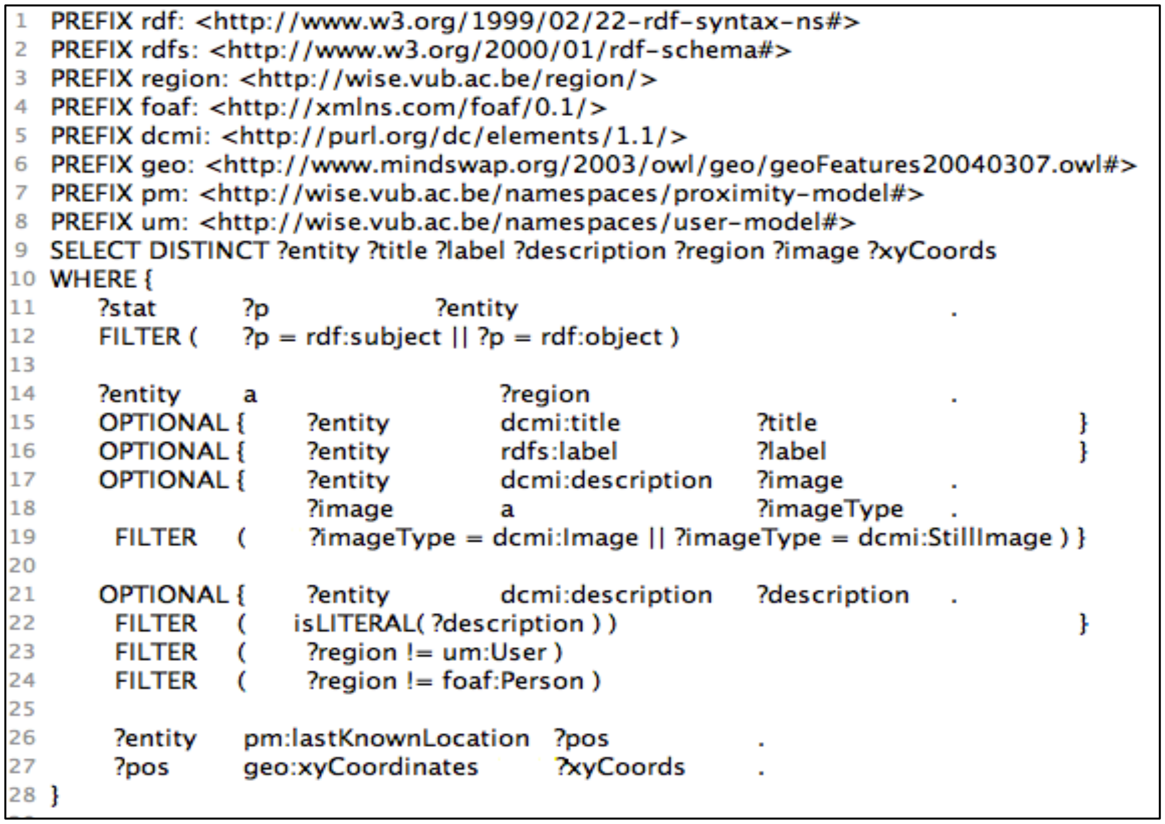
Figure 20 - Entity Data Type

The SCOUT framework (see section 2.4.3) is chosen to plug in the Context Repository component to obtain the user's context. The reason for that is because SCOUT is a lightweight, and scalable framework, and its simplicity, as it does not require a centralized server; instead, it accesses the physical entities' information directly on the web. Other context acquisition framework can be plugged in the Context Repository as well.

In this section, we will focus on how we implemented the three above-mentioned methods in the generic context interface to collect the physical entities via SCOUT framework. SCOUT employs its Environment Layer (see section 2.4.3.3) to provide the mobile application with an integrated view of the physical entities in the user's environment. As stated before, the Environment Layer supports the query service (by issuing the SPARQL queries) to access this data, and the notification service to notify about the changes in the user's environment.

Below, we show in detail how to use SPARQL queries to implement the three above methods in the generic context interface.

a) <u>Detect Entities:</u> detecting all entities in the user surroundings



```
PREFIX rdf: <http://www.w3.org/1999/02/22-rdf-syntax-ns#>
PREFIX rdfs: <http://www.w3.org/2000/01/rdf-schema#>
PREFIX region: <http://wise.vub.ac.be/region/>
PREFIX foaf: <http://xmlns.com/foaf/0.1/>
PREFIX dcmi: <http://purl.org/dc/elements/1.1/>
PREFIX geo: <http://www.mindswap.org/2003/owl/geo/geoFeatures20040307.owl#>
PREFIX pm: <http://wise.vub.ac.be/namespaces/proximity-model#>
PREFIX um: <http://wise.vub.ac.be/namespaces/user-model#>
SELECT DISTINCT ?entity ?title ?label ?description ?region ?image ?xyCoords
WHERE {
    ?stat        ?p              ?entity                                  .
    FILTER (     ?p = rdf:subject || ?p = rdf:object )

    ?entity      a                         ?region                        .
    OPTIONAL {       ?entity          dcmi:title           ?title                    }
    OPTIONAL {       ?entity          rdfs:label           ?label                    }
    OPTIONAL {       ?entity          dcmi:description     ?image          .
                     ?image           a                    ?imageType      .
      FILTER     (   ?imageType = dcmi:Image || ?imageType = dcmi:StillImage ) }

    OPTIONAL {       ?entity          dcmi:description     ?description    .
      FILTER     (  isLITERAL( ?description ) )                              }
      FILTER     (   ?region != um:User )
      FILTER     (   ?region != foaf:Person )

      ?entity    pm:lastKnownLocation  ?pos          .
      ?pos       geo:xyCoordinates     ?xyCoords     .
}
```

Figure 21 - SPARQL - Select All Detected Entities

An entity may contain a lot of information, such as title, label, a short description describing about the entity, a region that the entity is located in, an image to illustrate, the entity's coordination, and so on. An entity can have any of these properties, so we use OPTIONAL operator to identify. In the SPARQL above, an entity can either have a title, a label, a description or an image.

```
OPTIONAL (?entity dcmi:title ?title)
OPTIONAL (?entity dcmi:label ?label)
OPTIONAL (?entity dcmi:description ?description)
OPTIONAL (?image a ?imageType)
```

However, an entity must have a type, such as restaurant, university, coffee shop, etc. An entity also contains the position.

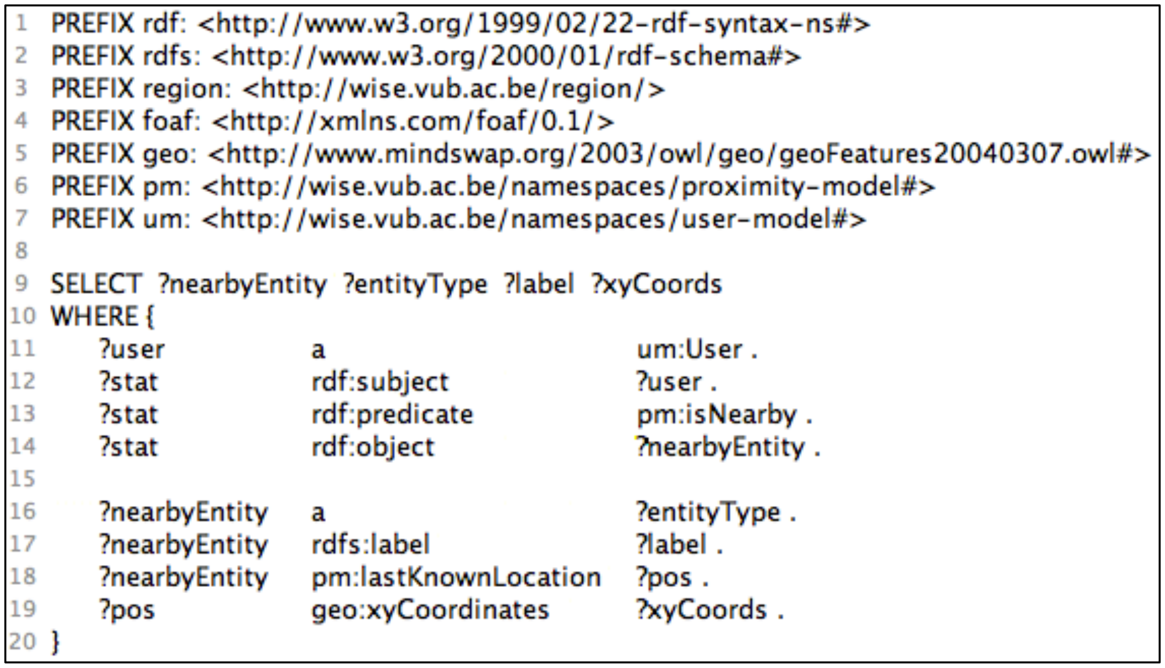
```
?entity a ?region
?entity pm:lastKnowLocation ?pos
```

We use FILTER to remove unnecessary results. Because the user can be considered as an entity, so to detect all entities in the user surroundings we should exclude the user entity. It's not accepted if the description is not literal.

```
FILTER (?region != um:User)
FILTER (?region != foaf:Person)
FILTER (isLITERAL(?description))
```

The information illustrating entities is stored in RDF files or webpages annotation, we assume that all entities consist images to describe themselves. These images will be displayed on the interface to represent entities. However, in the case, Internet sources do not contain images, the label, title or description can explain our entities.

b) **Nearby Entities**: find out all nearby entities in comparison with user location

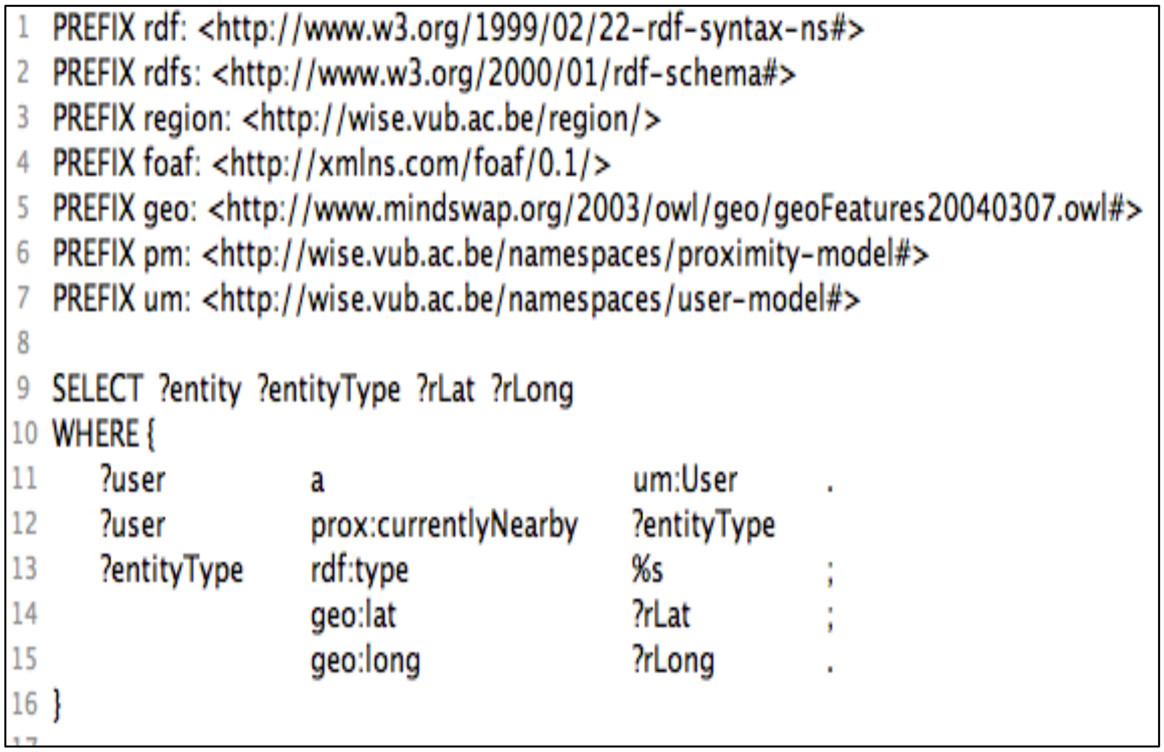


```
PREFIX rdf: <http://www.w3.org/1999/02/22-rdf-syntax-ns#>
PREFIX rdfs: <http://www.w3.org/2000/01/rdf-schema#>
PREFIX region: <http://wise.vub.ac.be/region/>
PREFIX foaf: <http://xmlns.com/foaf/0.1/>
PREFIX geo: <http://www.mindswap.org/2003/owl/geo/geoFeatures20040307.owl#>
PREFIX pm: <http://wise.vub.ac.be/namespaces/proximity-model#>
PREFIX um: <http://wise.vub.ac.be/namespaces/user-model#>

SELECT ?nearbyEntity ?entityType ?label ?xyCoords
WHERE {
    ?user           a                     um:User .
    ?stat           rdf:subject           ?user .
    ?stat           rdf:predicate         pm:isNearby .
    ?stat           rdf:object            ?nearbyEntity .

    ?nearbyEntity   a                     ?entityType .
    ?nearbyEntity   rdfs:label            ?label .
    ?nearbyEntity   pm:lastKnownLocation  ?pos .
    ?pos            geo:xyCoordinates     ?xyCoords .
}
```

Figure 22 - SPARQL - Select All Nearby Entities

The same to identify detected entities, the difference is in the predicate by utilizing the proximity model "isNearby".

c) **Search Entities by type**: search entities by type, such as find all restaurants in the user surroundings, find all the universities in the user surroundings.



```
PREFIX rdf: <http://www.w3.org/1999/02/22-rdf-syntax-ns#>
PREFIX rdfs: <http://www.w3.org/2000/01/rdf-schema#>
PREFIX region: <http://wise.vub.ac.be/region/>
PREFIX foaf: <http://xmlns.com/foaf/0.1/>
PREFIX geo: <http://www.mindswap.org/2003/owl/geo/geoFeatures20040307.owl#>
PREFIX pm: <http://wise.vub.ac.be/namespaces/proximity-model#>
PREFIX um: <http://wise.vub.ac.be/namespaces/user-model#>

SELECT ?entity ?entityType ?rLat ?rLong
WHERE {
    ?user          a                     um:User        .
    ?user          prox:currentlyNearby  ?entityType
    ?entityType    rdf:type              %s             ;
                   geo:lat               ?rLat          ;
                   geo:long              ?rLong         .
}
```

Figure 23 - SPARQL - Search Nearby Entity By Type

This query first identify the user, and then obtains entities currently nearby by utilizing the proximity model (prox:currentlyNearby). Afterward, it checks whether the return nearby entities are the type that is passed through the query via "%s" parameter.

```
?entityType    rdf:type       %s
```

Finally, the ContextManager class supports the auto-posting and semi-auto posting features. By storing the hometown user's location, and exploiting the amount of time the user is at a specific entity, the ContextManager class enables notifying when the user is abroad and when the time the user is at a specific place is over a predetermined amount of hours, whereby the entity's name can be automatically posted online. On the other hand, the ContextManager stores all entities that the user visits during a day, whereby the user can select the entities to share.

4.3.3 Filter Process

The FilterProcess class is responsible for calculating the context-relevance of data (e.g., a group, an event, a post, a photo), obtained from social networks. For example, there are 2 events happening at the same time at the user's current location. Assume that football is the user's favorite sport, so the football event would be more relevant than another one.

The filter process is implemented by the ContextMatching function that takes the following parameters:

- **Social media data** obtained from Social Network Layer (e.g., a group, an event, a photo, a post).

- **A nearby entity** obtained from Context Repository that the user is currently interested in.

- **The social user's profile** obtained from Social Network Layer (e.g., user's favorite movies, places, sports, etc.)

The result that the function returns is a natural integer number, which represents the degree of similarity. The degree of similarity is measured by summing up the following points.

The User's Context		
Location	**Description**	**Point**
	The distance is within a radius of 1 km	3
	The distance is within a radius of 2 km	2
	The distance is within a radius of 5 km	1
	The distance is over a radius of 5 km	0
	The distance here is the distance between the user's current location and the location of the social media data (e.g., a photo, a group, an event)	
Time		
	The social media data was created today	7
	The social media data was created yesterday	6
	The social media data was created 2 days ago	5
	The social media data was created 3 days ago	4
	The social media data was created 4 days ago	3
	The social media data was created 5 days ago	2
	The social media data was created 6 days ago	1
	The social media data was created a week ago	0
Nearby Entity		
	Each field of the social media data contains the nearby entity's name	1

Figure 24 - Context Matching - The User's Context

Based on the user's current context, the Context Matching function will calculate the degree of similarity in terms of points. In particular, the function will identify the distance between the user's current location and the location where the social media data is posted. The closer the distance is, the higher the point the relevance receives. On the other hand, the Context Matching function evaluates the social media data based on time. It means that the newest social media data will receive a higher point than an old one. In addition, the more frequently the nearby entity's name appears in the social media data, the higher the point the relevance will get.

The User's Profile		
Relationship	**Description**	**Point**
	The author of the social media data is in the user's family group	3
	The author of the social media data is in the user's close friends group	2
	The author of the social media data is in the user's friend list	1
	The author of the social media data has no relationship with the user	0
Preference		
	Any field of the social media data contains any user's favorite movie's name	1
	Any field of the social media data contains any user's favorite book's name	1
	Any field of the social media data contains any user's favorite music's name	1
	Any field of the social media data contains any user's favorite sport's name	1
	Any field of the social media data contains any user's favorite place's name	1
	Any field of the social media data contains any user's favorite TV Show's name	1
	Any field of the social media data contains any user's workplace's name	1
	Any field of the social media data contains any user's school's (or university's) name	1

Figure 25 - The Context Matching - The User's Profile

The Context Matching function exploits the user's profile information to identify the degree of similarity. The function will give points based on the relationship between the user and the author of the social media data. The closer the relationship is, the more points the relevance gets. On the other hand, the function gives points based on the user's preference. The more frequently the user's favorite activities or hobbies (e.g., sports, music, movies) appear in the social media data, the more points the relevance receives.

The Filter Process returns the degree of similarity, and the User Interface Layer will arrange the social media data based on this degree. The higher point the degree returns, the more relevant the data is. The more relevant data will appear prior to the less relevant data.

4.4 User Interface Layer

The User Interface Layer (UI) is the space where interactions between the user and mobile device occur. Our aim is to separate the application logic and the User Interface concerns so that changes to the user interface will not affect the data handling, and the data handling part (e.g., Data Processing Layer, Social Network Layer) can be reorganized without changing the User Interface. Specifically, the User Interface does not depend on either from which social network returns the social media data, or how many social networks are plugged into our framework. The User Interface only depends on the common data type (see section 4.2.1) found in social networks. The communication between the User Interface layer and the layers below is asynchronous. The UI does not have to wait for the data coming, but is notified whenever new data is available.

The User Interface Layer contains the following features:

- Displaying a list of nearby physical entities. In this list, we can search entities by type (e.g., restaurants, university, sport, etc.). Please see section 5.11 for more details.

- Displaying the user's current location in comparison with other nearby entities on map (see section 5.12).

- Displaying a list of requests for finding social media relevant-context data. This includes finding the user’s friends who work, or live at a particular place, who attends events that take place in the user's surroundings, and so on (see section 5.4)

- Displaying the common data type returned by the bottom layer (the Data Processing Layer). This includes the photo, group, events, and etc. data types.

Please see sections 5.5, 5.6, 5.7, 5.8, 5.9, and 5.10 for more details. The data will be arranged based on the degree of similarity.

- Displaying a preference setting, allowing user auto-posting the context. This includes the scenarios when the user travel abroad, or when the user stays at a specific context longer than a given amount of time (see section 5.14)

- Displaying the history context, allowing the user semi-auto posting the context. The history screen shows all contexts that the user has visited during the day. At a specific time of day, i.e. at the end of day, the user can decide if any context the user finds interesting, he or she can rate it, and share it on social networks. That is the way we implement the semi-auto posting feature in our application (see section 5.13)

The next section deals with the development of an Android application based on our framework. Via this application, we also see how the User Interface may look like.

5 Proof-of-Concept Mobile App

In this section, we describe a mobile application built upon the above framework to access three social networks (Facebook, Google Plus, and Twitter) to find the relevant-context social media data. Firstly, we build a shared data type among these social networks, and then built a set of shared methods accordingly to access these data found in social networks. Afterwards, we will show off the result of finding the relevant-context data from these social networks. Other features (such as, auto-posting mechanism, semi-auto posting mechanism, searching the user's context by type, displaying the map location) will be included as well.

5.1 Shared Data Types

Under the scope of the application we built, we integrated three social networks platforms, Facebook, Twitter, and Google Plus, and we identified a main shared vocabulary between them as follow:

Facebook	**Google Plus**	**Twitter**	**Our System**	**Description**
New Feed	Stream	Updates	Stream	New updated information visible to the user
Wall	Profile	TimeLine	Profile	All information appears on the user's home page
Friends	Followers & Circles	Following & Followers	Friends	Information about people in the user's contact
Events	Events	None	Events	Information about an event, including the location, event name, and which invitees plan to attend
Groups	Communities	None	Groups	Information about a group, including group name, group description, and the owner of the group
Notifications	Notifications		Notifications	Quick updates about friends' action that most affect the user
Posts	Shares	Tweets	Posts	An individual entry or status update that a user shares on social network page

Likes	+1's	Favorites	Likes	An expression shows the user's preference
Photo	Photo	Media	Photo	Information about an individual photo, including image, the associated information like the tagged user and their position in this photo
Hashtag	Hashtag	Hashtag	Hashtag	A word or a phrase after the symbol #, indicating an event or a topic that people are discussing about
Album	Album	None	Album	Information about an album of photos, including the album description, the location of the album, and the number of photos in this album
Check-ins	Check-ins	Geo-Place	Check-ins	Representation of a single visit to a location, including the place name, the message the user added to, all comments, and the users the author tagged in the check-in
Place	Place	Place	Place	Information about the social network page that represents a specific location
User	User	User	User	Information about a specific person
Comments	Comments	Retweets	Comments	A text that people use to answer on a social network object, like a post, a check-in, or activities.

Figure 26 - Shared Vocabulary

5.2 Shared Methods

Besides a shared vocabulary, we built a set of common methods shared among Facebook, Twitter, and Google Plus as follow:

Shared Methods	Description
Finding People	Finding friends whose activities are related to the user's current location (e.g., living, working, studying nearby)
Finding Check-Ins	Finding friends who visited a place in a specific range of user current location (for example, in a range of 100 meters diameter)
Finding Events	Finding all events that belongs to a specific entity, such as building, company, university that are located in the user's surroundings, or events that take place nearby the user's current location
Finding Groups	Finding groups with activities that are related to the user's location, such as the group belongs to a particular place (e.g., building) or organizations (e.g., company, university)
Finding Photos	Finding all photos, and albums that the user's friends have taken of places or things in the user's current surroundings
Finding Posts	Finding all public posts that contains information about a specific entity, such as tourist attraction, organization, university that are located in the user's surroundings

Figure 27 - Shared Methods

The next sections will describe how we created these above shared methods to access resources on Facebook, Google Plus, and the Twitter Platform.

5.3 Account Screen

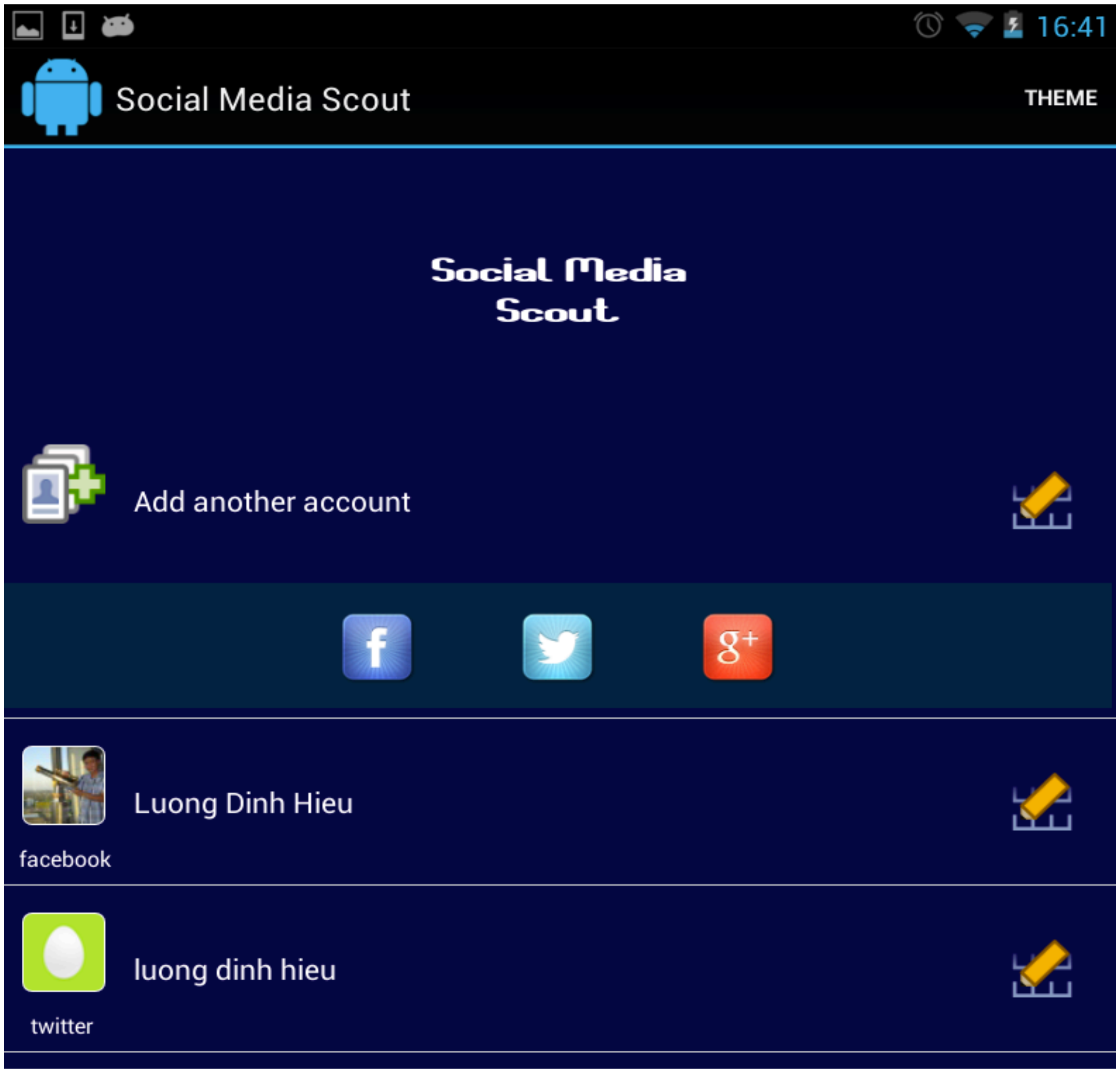


Figure 28 - The Account Screen - integrate Facebook, Twitter and Google Plus

The Account Screen is the first screen when the application is launched (see Figure 25). The Account Screen allows integrating the three most popular social networks at once: Facebook, Twitter, and Google Plus. By tapping on the button "Add another account", we can integrate these social networks into our application.

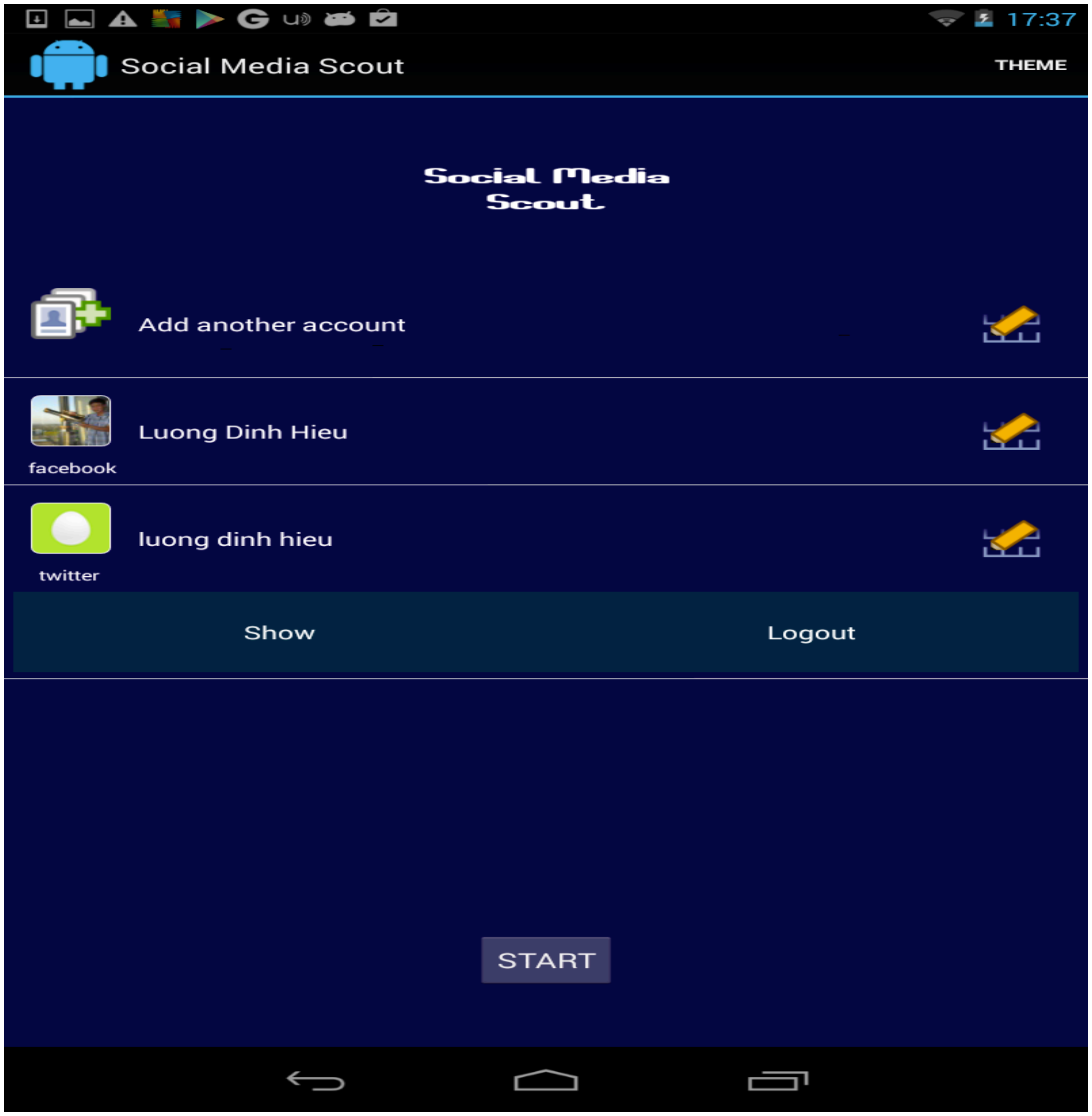


Figure 29 - The Account Screen - Showing Profile, Logout and Start Application

The next rows (see Figure 26) show the information of the current integrated social networks (e.g., the social network's name, the user's full name, the user's avatar picture). Tapping on the "Edit" buttons (located at the right side of each row), we can obtain more details of a specific profile avatar (tapping on button "Show") and we can logout from a specific social network (tapping on button "Logout").

The user starts our application by tapping on the button "START", which is located on the bottom of this screen.

5.4 Home Screen

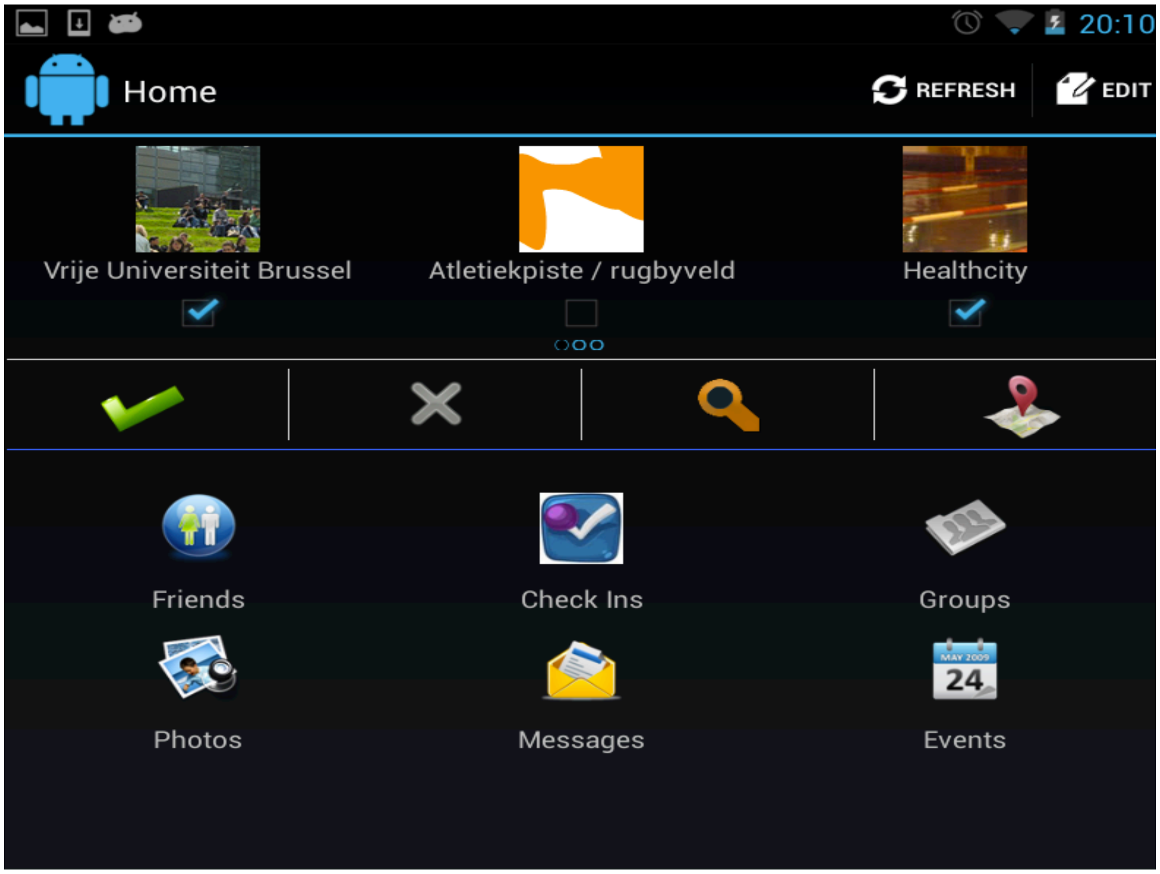


Figure 30 - The Home Screen

The Home Screen (see Figure 27) enables the user to find the relevant-context social media data (e.g., finding events, groups, check-ins related to the physical entities in the user's surroundings). The Home Screen consists of three parts:

Part1:

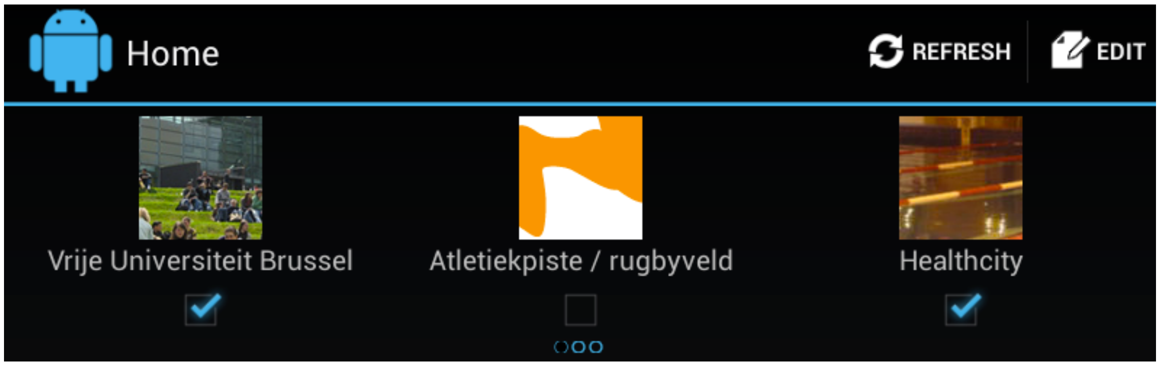


Figure 31 - The Home Screen - Collecting Physical Entities

This part is for presenting the real physical entities detected in the user's surroundings. The listing 5.4 enumerates the detected entities in VUB University area: Health City, Complex Building, Opinio Coffee Shop, Student Restaurant, and so on. All detected entities are stored in a list view, and the user can scroll horizontally to see the rest of entities. Importantly, the user can select some entities that he is interested in to later extract later the meaningful information related to these selected entities from social networks (e.g., finding events that take place there, finding pictures that the user's friends took there).

Part2:

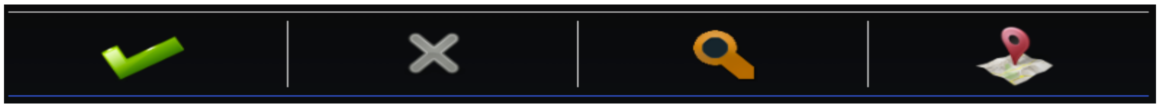


Figure 32 - The Home Screen - Functions related to the physical entities

This part shows some functions to related to the detected physical entities as specified follow:

Functions	Description
✓	Selecting all detected entities in a page of the list view.
✕	De-selecting all selected entities in a page of the list view.
🔍	Searching detected entities by type (e.g., restaurant, coffee shop, etc.). This function directly links to the Search Screen (see section 5.11).
📍	Showing all detected entities and the user's current location on map. This function directly links to the Map Screen (see section 5.12)

Figure 33 - The Home Screen - Functions related to the physical entities

Part3:

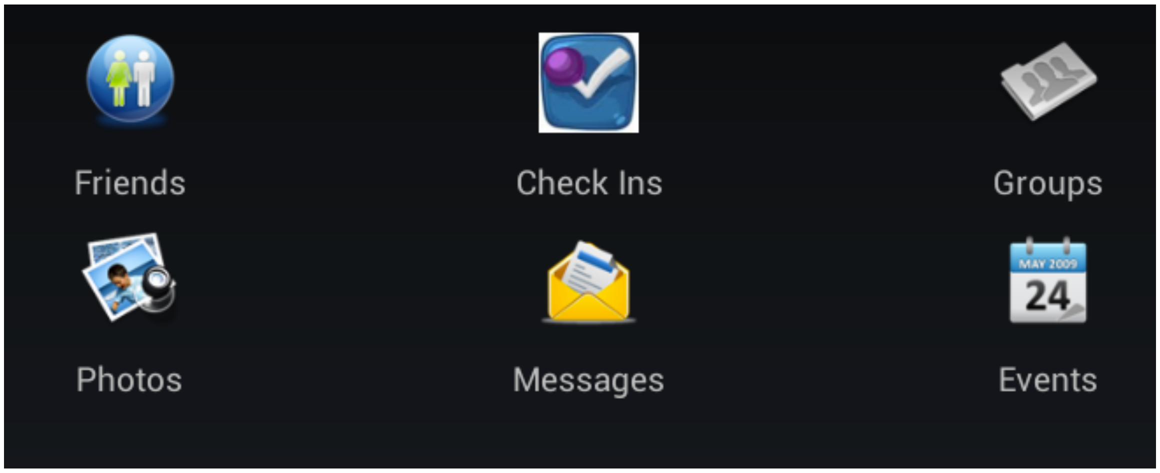


Figure 34 - The Home Screen - Shared Methods

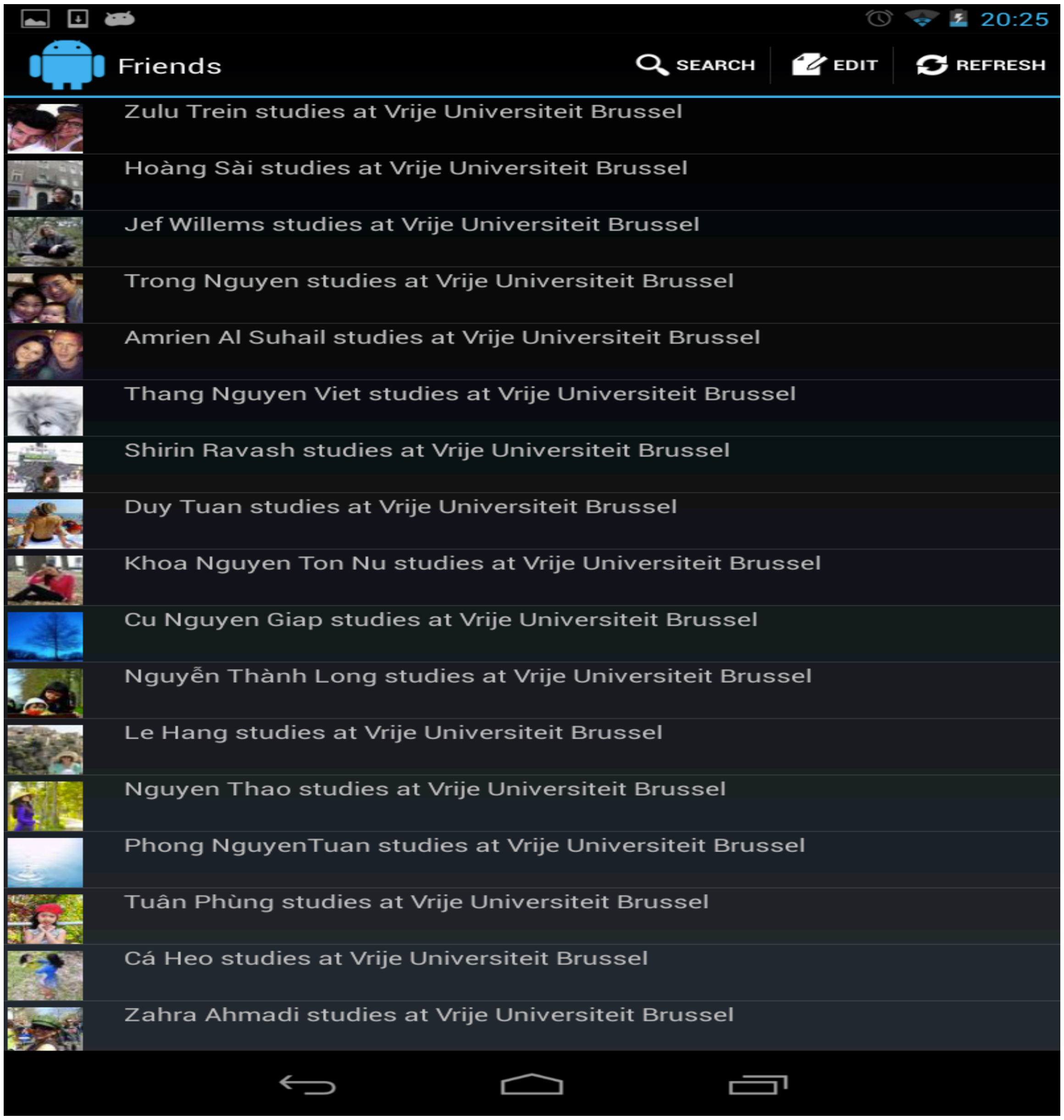
This part is for representing our set of shared methods (see section 5.2) to obtain the relevant-context social media data.

Shared Method	Description
👫	Finding friends whose activities are related to the user's current location (e.g., living, working, studying nearby). This method directly links to the Friends Screen (see section 5.5).
✔	Finding friends who visited a place in a specific range of user current location (for example, in a range of 100 meters diameter). This method directly links to the Check-ins Screen (see section 5.6).
👥	Finding groups with activities that are related to the user's location, such as the group belongs to a particular place (e.g., building) or organizations (e.g., company, university). This method directly links to the Groups Screen (see section 5.7).
🖼	Finding all photos, and albums that the user's friends have taken of places or things in the user's current surroundings. This method directly links to the Photos Screen (see section 5.8).
✉	Finding all public posts that contains information about a specific entity, such as tourist attraction, organization, university that are located in the user's surroundings. This method directly links to the Posts Screen (see section 5.9).
📅	Finding all events that belongs to a specific entity, such as building, company, university that are located in the user's surroundings, or events that take place nearby the user's current location. This method directly links to the Events Screen (see section 5.10).

Figure 35 - The Home Screen - Shared Methods Description

As an example we could imagine is if you want to know about what people talked about the service of the Health City at VUB campus, or what events of VUB you can enroll, you just select the VUB entity, and the Health City entity in the first part, and then tap on the shared methods below related to finding posts and events.

5.5 Friends Screen

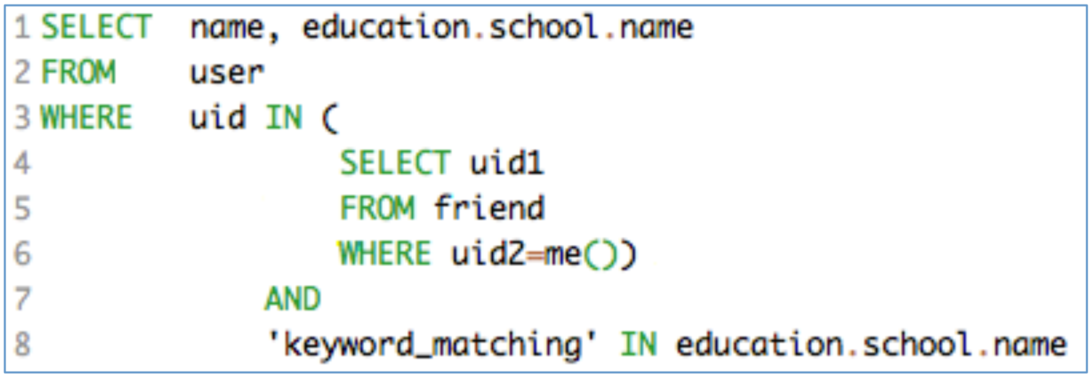


Zulu Trein studies at Vrije Universiteit Brussel
Hoàng Sài studies at Vrije Universiteit Brussel
Jef Willems studies at Vrije Universiteit Brussel
Trong Nguyen studies at Vrije Universiteit Brussel
Amrien Al Suhail studies at Vrije Universiteit Brussel
Thang Nguyen Viet studies at Vrije Universiteit Brussel
Shirin Ravash studies at Vrije Universiteit Brussel
Duy Tuan studies at Vrije Universiteit Brussel
Khoa Nguyen Ton Nu studies at Vrije Universiteit Brussel
Cu Nguyen Giap studies at Vrije Universiteit Brussel
Nguyễn Thành Long studies at Vrije Universiteit Brussel
Le Hang studies at Vrije Universiteit Brussel
Nguyen Thao studies at Vrije Universiteit Brussel
Phong NguyenTuan studies at Vrije Universiteit Brussel
Tuân Phùng studies at Vrije Universiteit Brussel
Cá Heo studies at Vrije Universiteit Brussel
Zahra Ahmadi studies at Vrije Universiteit Brussel

Figure 36 - Friends Screen

The Friends Screen is to find the user's friends whose activities are related to the user's current location (e.g., living, working, studying nearby). The Figure 32 illustrates a scenario where we are crossing the VUB University and we would like to know who are our friends that are studying at, working at, or living nearby the VUB University. Below, we explain how we can achieve this result by accessing the Facebook (see section 4.2.2) and Twitter Search API (4.2.3).

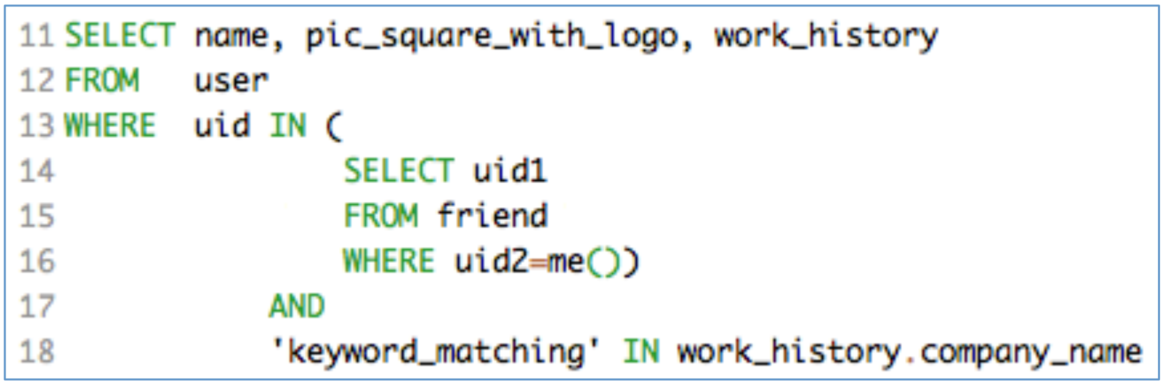
For Facebook, we use both Graph Search API and FQL query to implement this request.



```
SELECT  name, education.school.name
FROM    user
WHERE   uid IN (
                SELECT uid1
                FROM friend
                WHERE uid2=me())
            AND
            'keyword_matching' IN education.school.name
```

Figure 37 - FQL to find friends' education matching user's context

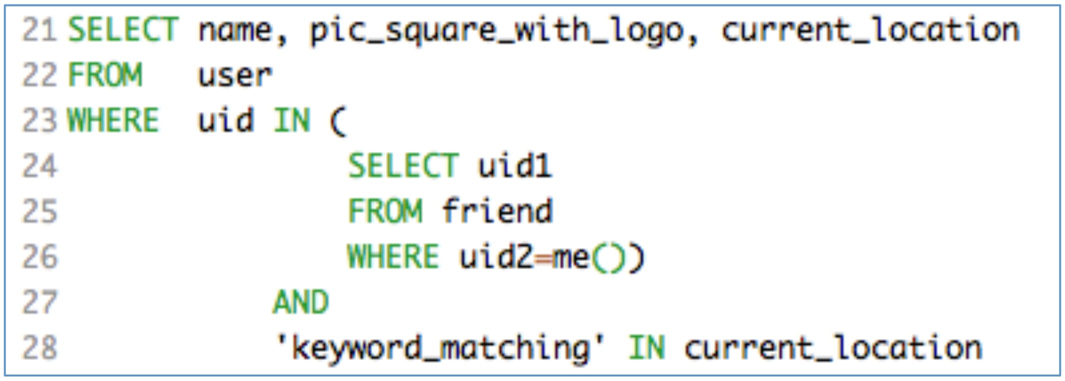
This FQL query finds the user's friends who study at a place with the name is the same or contains the keyword. The keyword here is the name or the abbreviation name of the place.



```
SELECT name, pic_square_with_logo, work_history
FROM   user
WHERE  uid IN (
               SELECT uid1
               FROM friend
               WHERE uid2=me())
           AND
           'keyword_matching' IN work_history.company_name
```

Figure 38 - FQL to find friends' work place matching user's context

This FQL query finds the user's friends who work at a place with the name is the same or contains the user's context keyword.



```
SELECT name, pic_square_with_logo, current_location
FROM   user
WHERE  uid IN (
               SELECT uid1
               FROM friend
               WHERE uid2=me())
           AND
           'keyword_matching' IN current_location
```

Figure 39 - FQL to find friends' living place matching user's context

This FQL query finds the user's friends who live at a place with the name is the same or contains the user's context keyword.

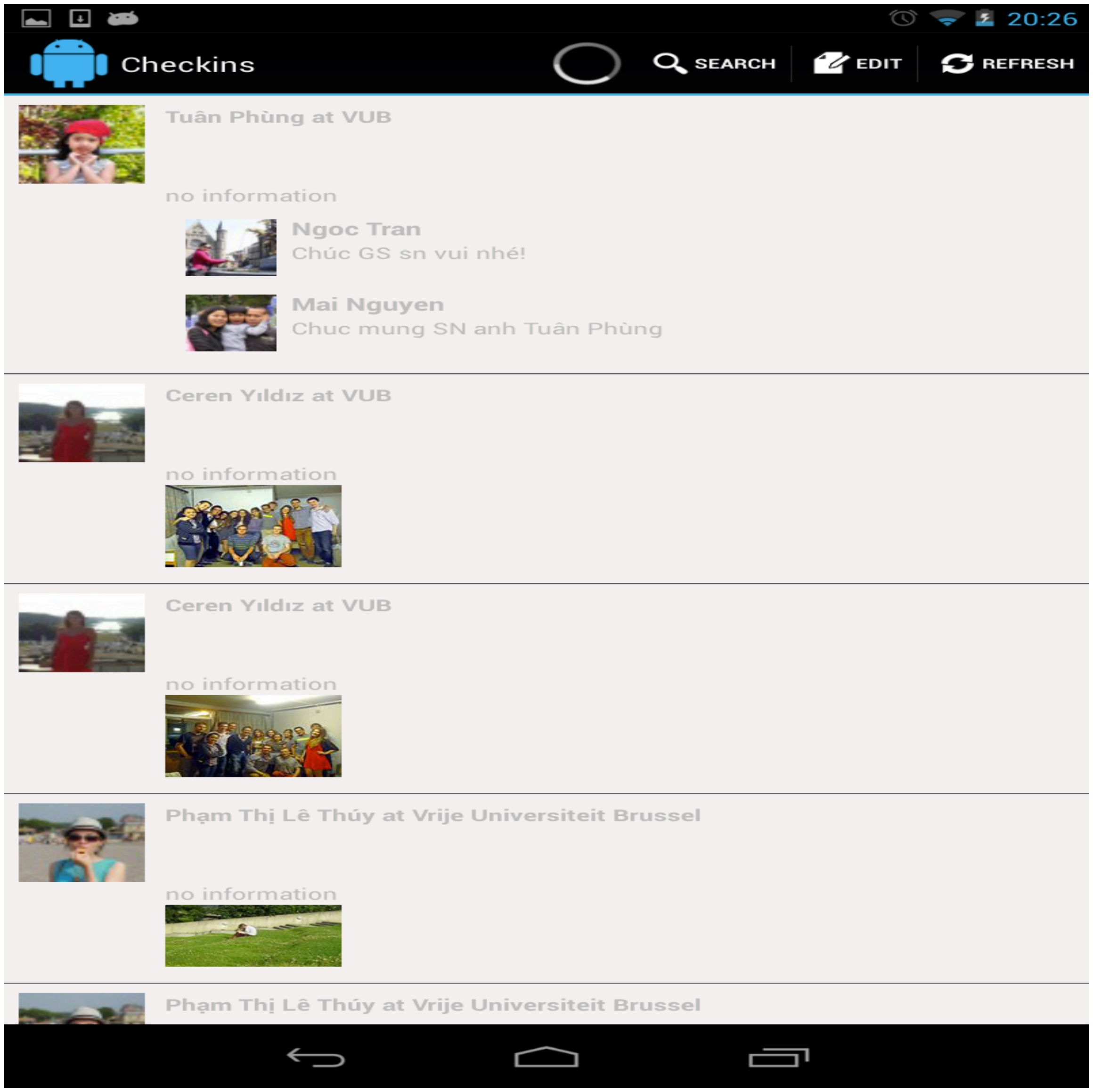
Using Twitter API version 1.1, to find all friends who live near the location of a specific context.

- GET friends/list [22] - return a collection of user object for every user the specified user is following (otherwise known as their friends)

User/location [23] - extract location of a user.

[22] https://dev.twitter.com/docs/api/1.1/get/friends/list
[23] https://dev.twitter.com/docs/platform-objects/users

5.6 Check-ins Screen



Tuân Phùng at VUB

no information

Ngoc Tran
Chúc GS sn vui nhé!

Mai Nguyen
Chuc mung SN anh Tuân Phùng

Ceren Yıldız at VUB

no information

Ceren Yıldız at VUB

no information

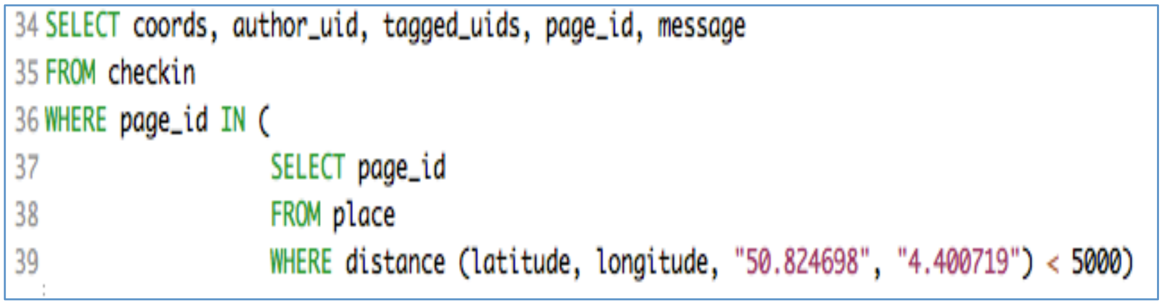
Phạm Thị Lê Thúy at Vrije Universiteit Brussel

no information

Phạm Thị Lê Thúy at Vrije Universiteit Brussel

Figure 40 - The Check-Ins Screen

The Check-ins Screen is to find friends who visited a place in a specific range of the user's current location (for example, in a range of 100 meters diameter). The Check-Ins information includes the photo taken at that place, the post and the associated comments. The figure 36 illustrates the user's friends who have visited the VUB University. Below, we explain how we can achieve this result by accessing the Facebook (see section 4.2.2) and Twitter Search API (4.2.3).

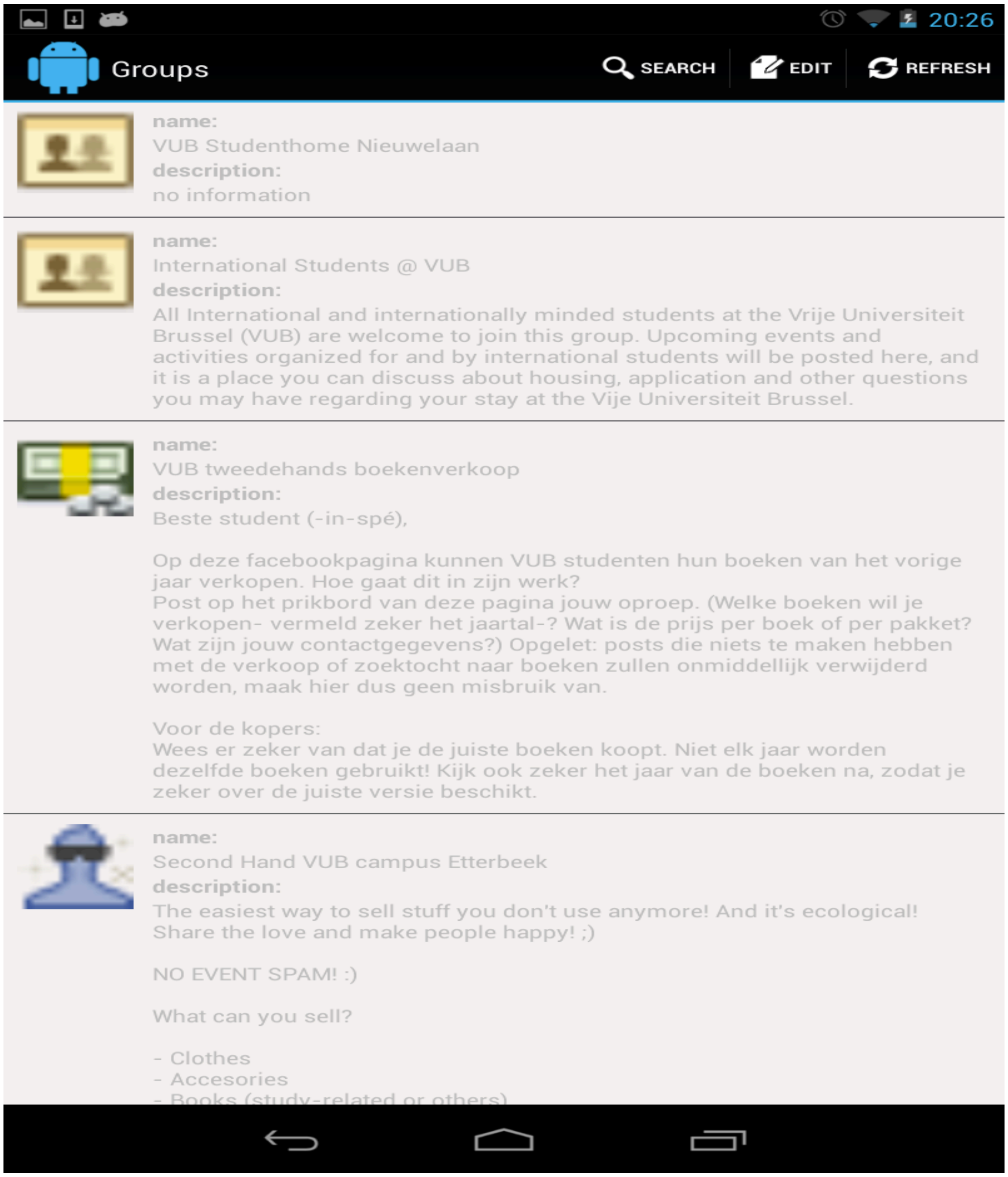


```
SELECT coords, author_uid, tagged_uids, page_id, message
FROM checkin
WHERE page_id IN (
                  SELECT page_id
                  FROM place
                  WHERE distance (latitude, longitude, "50.824698", "4.400719") < 5000)
```

Figure 41 - FQL to find friends' check-ins in a specific location

This service is to get check-ins of the user's friends in a specific range of user current location (for example, in a range of 5000 meters diameter) and match the user's context. This query collects all check-ins in a specific location with diameter of 5 kilometers. The raw data will be passed to the filter process to find the most relevant-context data.

5.7 Group Screen



name:
VUB Studenthome Nieuwelaan
description:
no information

name:
International Students @ VUB
description:
All International and internationally minded students at the Vrije Universiteit Brussel (VUB) are welcome to join this group. Upcoming events and activities organized for and by international students will be posted here, and it is a place you can discuss about housing, application and other questions you may have regarding your stay at the Vije Universiteit Brussel.

name:
VUB tweedehands boekenverkoop
description:
Beste student (-in-spé),

Op deze facebookpagina kunnen VUB studenten hun boeken van het vorige jaar verkopen. Hoe gaat dit in zijn werk?
Post op het prikbord van deze pagina jouw oproep. (Welke boeken wil je verkopen- vermeld zeker het jaartal-? Wat is de prijs per boek of per pakket? Wat zijn jouw contactgegevens?) Opgelet: posts die niets te maken hebben met de verkoop of zoektocht naar boeken zullen onmiddellijk verwijderd worden, maak hier dus geen misbruik van.

Voor de kopers:
Wees er zeker van dat je de juiste boeken koopt. Niet elk jaar worden dezelfde boeken gebruikt! Kijk ook zeker het jaar van de boeken na, zodat je zeker over de juiste versie beschikt.

name:
Second Hand VUB campus Etterbeek
description:
The easiest way to sell stuff you don't use anymore! And it's ecological! Share the love and make people happy! ;)

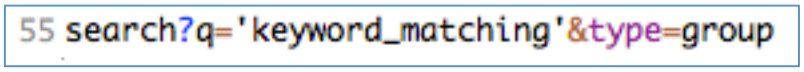
NO EVENT SPAM! :)

What can you sell?

- Clothes
- Accesories
- Books (study-related or others)

Figure 42 - The Group Screen

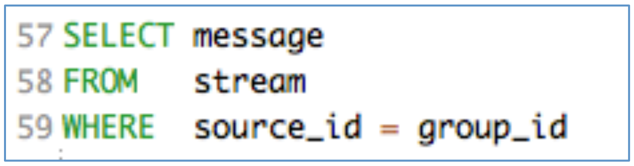
The Group Screen is to find groups with activities that are related to the user's location, such as the group belongs to a particular place (e.g., building) or organizations (e.g., company, university). The listing 5.9 illustrates all the groups with activities that are related to the VUB University (e.g., the VUB Student Home Nieuwelaan group, the VUB international students group, the second-hand stuffs for VUB students, etc.). Below, we explain how we can achieve this result by accessing the Facebook (see section 4.2.2) and Twitter Search API (4.2.3).



```
search?q='keyword_matching'&type=group
```

Figure 43 - Facebook Graph Search to find all groups related to user's context

This query is to get all groups that match the user context, or their activities take place in the user's current neighborhood.

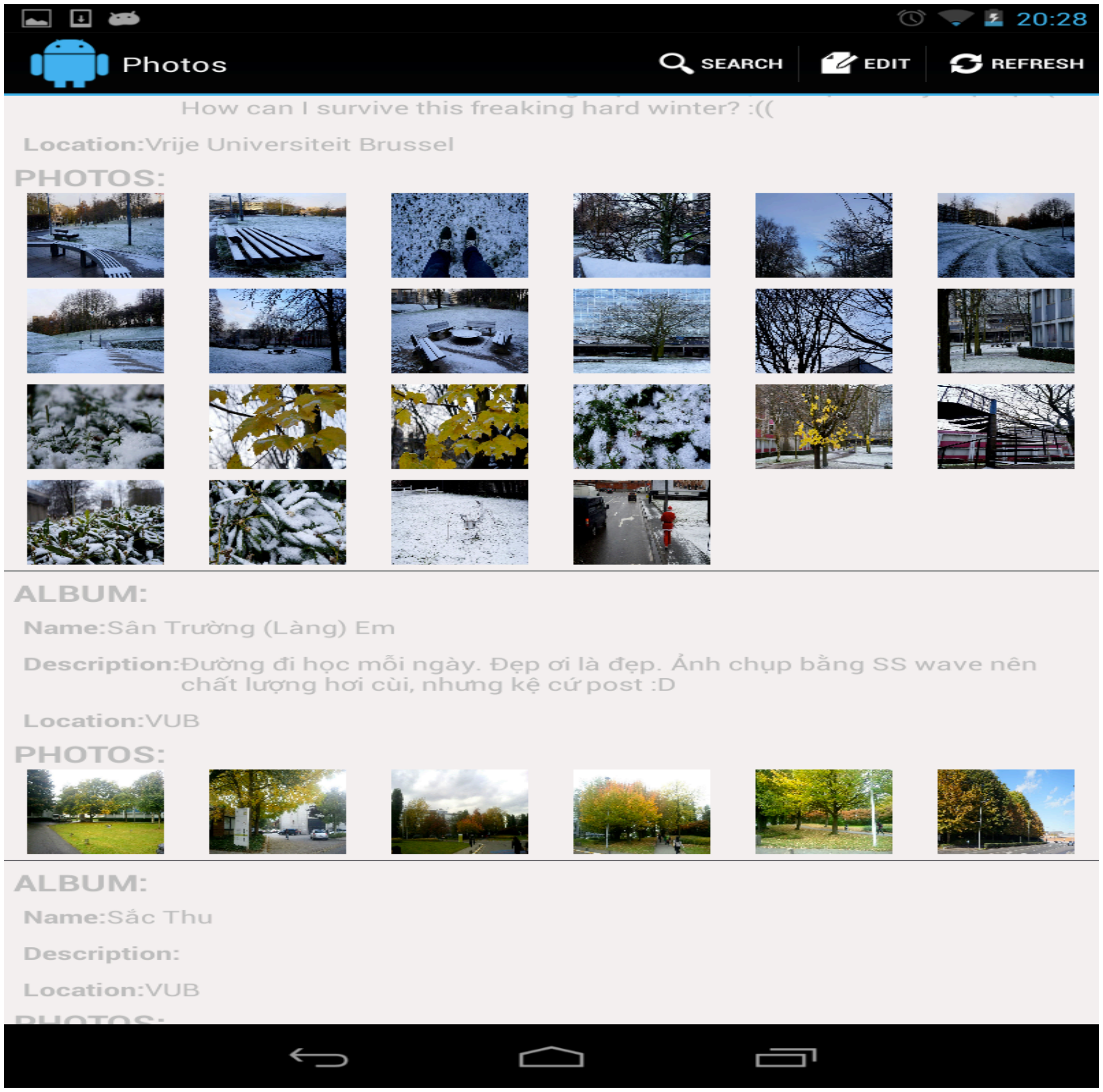


```
SELECT message
FROM   stream
WHERE  source_id = group_id
```

Figure 44 - FQL to get all recent news from a specific group

This query gets all recent news from a specific group by passing group id.

5.8 Photos Screen



How can I survive this freaking hard winter? :((

Location:Vrije Universiteit Brussel

PHOTOS:

ALBUM:

Name:Sân Trường (Làng) Em

Description:Đường đi học mỗi ngày. Đẹp ơi là đẹp. Ảnh chụp bằng SS wave nên chất lượng hơi cùi, nhưng kệ cứ post :D

Location:VUB

PHOTOS:

ALBUM:

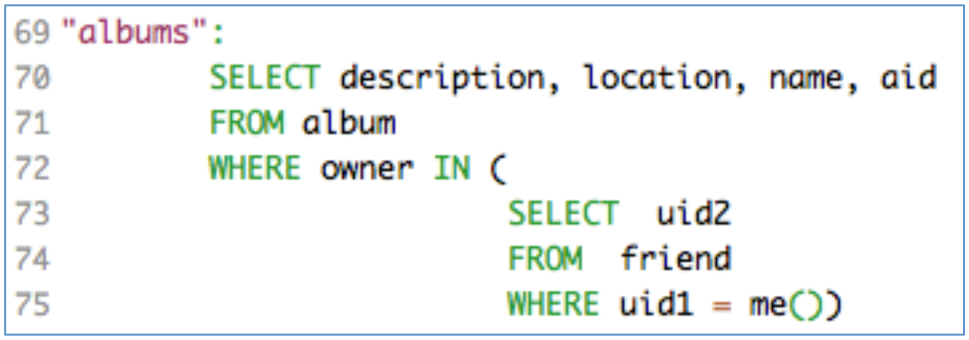
Name:Sắc Thu

Description:

Location:VUB

Figure 45 - The Photos Screen

The Photos Screen is to find all photos, and albums that the user's friends have taken of places or things in the user's current surroundings. The figure 40 illustrates all pictures taken place at VUB campus by the user's friends. Below, we explain how we can achieve this result by accessing the Facebook (see section 4.2.2) and Twitter Search API (4.2.3).



```
"albums":
        SELECT description, location, name, aid
        FROM album
        WHERE owner IN (
                        SELECT  uid2
                        FROM  friend
                        WHERE uid1 = me())
```

Figure 46 - FQL to get information about the user's friends' albums

This query finds out all albums of the user's friends.

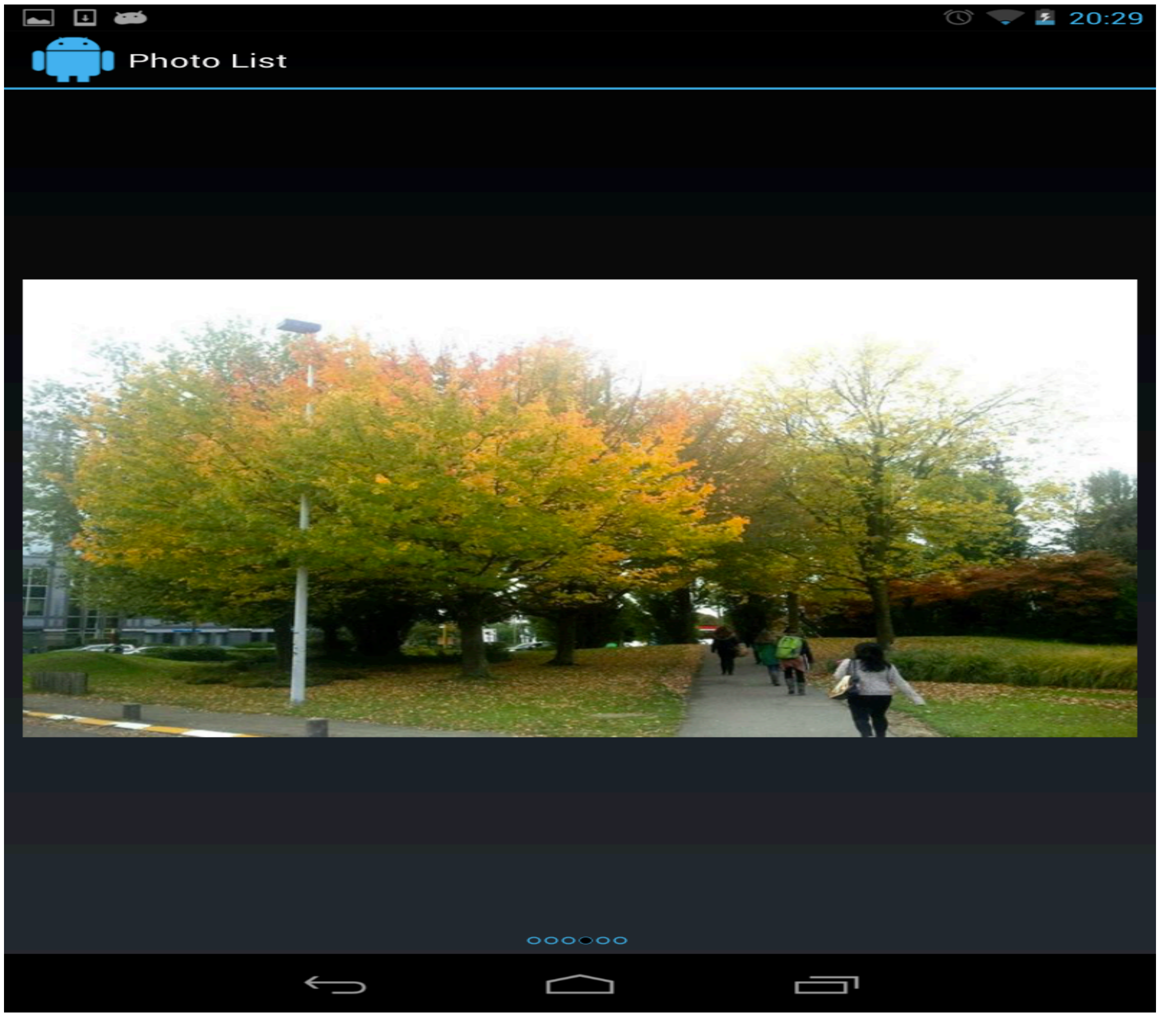
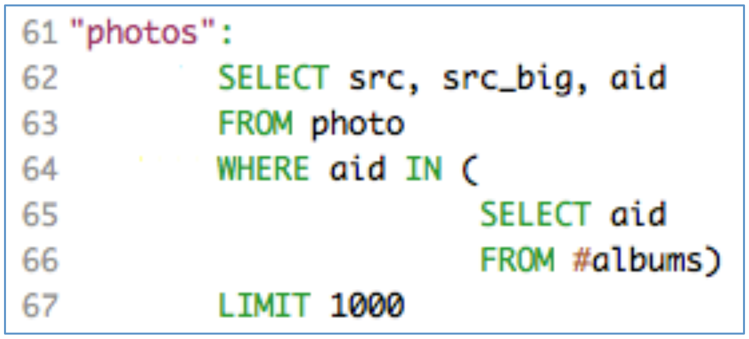


Figure 47 - The Photo List



```
"photos":
        SELECT src, src_big, aid
        FROM photo
        WHERE aid IN (
                      SELECT aid
                      FROM #albums)
        LIMIT 1000
```

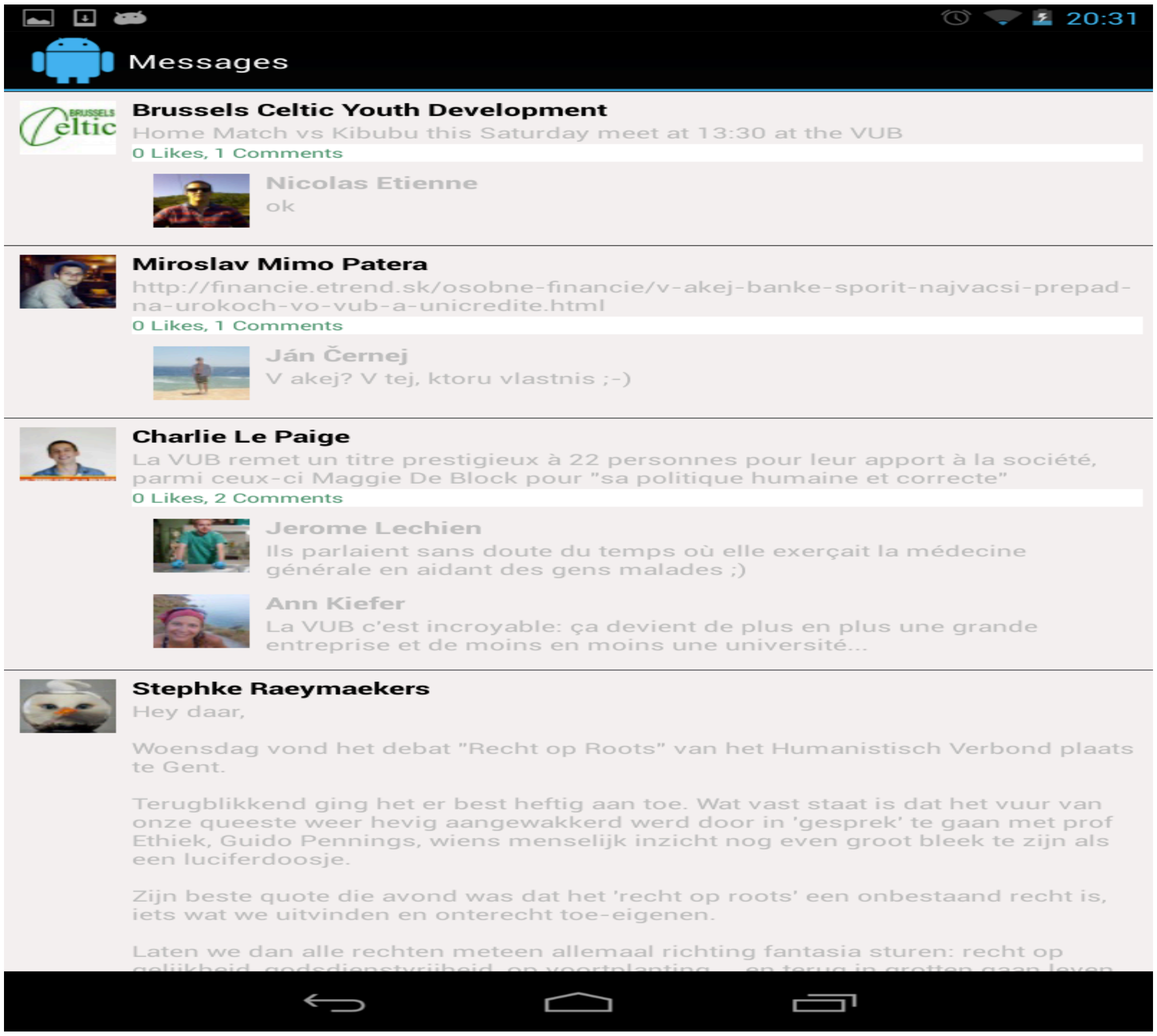
Figure 48 - FQL to get all photos in a specific album

This query finds all photos of a specific album. The maximum in the album is limited to 1000 photos. We use the multi-query FQL in this situation by passing the album query into photo query.

We pass the raw data (user's friends' photos) to the filter process to find the most relevant photos that match the user's context.

Using Twitter Search API to find tweets which match context and contain images, by including "filter:images", searching by topic #keyword_context

5.9 Posts Screen



Brussels Celtic Youth Development
Home Match vs Kibubu this Saturday meet at 13:30 at the VUB
0 Likes, 1 Comments

Nicolas Etienne
ok

Miroslav Mimo Patera
http://financie.etrend.sk/osobne-financie/v-akej-banke-sporit-najvacsi-prepad-na-urokoch-vo-vub-a-unicredite.html
0 Likes, 1 Comments

Ján Černej
V akej? V tej, ktoru vlastnis ;-)

Charlie Le Paige
La VUB remet un titre prestigieux à 22 personnes pour leur apport à la société, parmi ceux-ci Maggie De Block pour "sa politique humaine et correcte"
0 Likes, 2 Comments

Jerome Lechien
Ils parlaient sans doute du temps où elle exerçait la médecine générale en aidant des gens malades ;)

Ann Kiefer
La VUB c'est incroyable: ça devient de plus en plus une grande entreprise et de moins en moins une université...

Stephke Raeymaekers
Hey daar,

Woensdag vond het debat "Recht op Roots" van het Humanistisch Verbond plaats te Gent.

Terugblikkend ging het er best heftig aan toe. Wat vast staat is dat het vuur van onze queeste weer hevig aangewakkerd werd door in 'gesprek' te gaan met prof Ethiek, Guido Pennings, wiens menselijk inzicht nog even groot bleek te zijn als een luciferdoosje.

Zijn beste quote die avond was dat het 'recht op roots' een onbestaand recht is, iets wat we uitvinden en onterecht toe-eigenen.

Laten we dan alle rechten meteen allemaal richting fantasia sturen: recht op

Figure 49 - The Posts Screen

The Posts Screen is to finding all public posts that contains information about a specific entity, such as tourist attraction, organization, university that are located in the user's surroundings. The listing 5.12 illustrates the public posts about VUB University. Below, we explain how come we can achieve this result by accessing the Facebook (see section 4.2.2) and Twitter Search API (4.2.3).

```
search?q='keyword_matching'&type=post
```

Figure 50 - Facebook Graph Search to find all public posts related to context

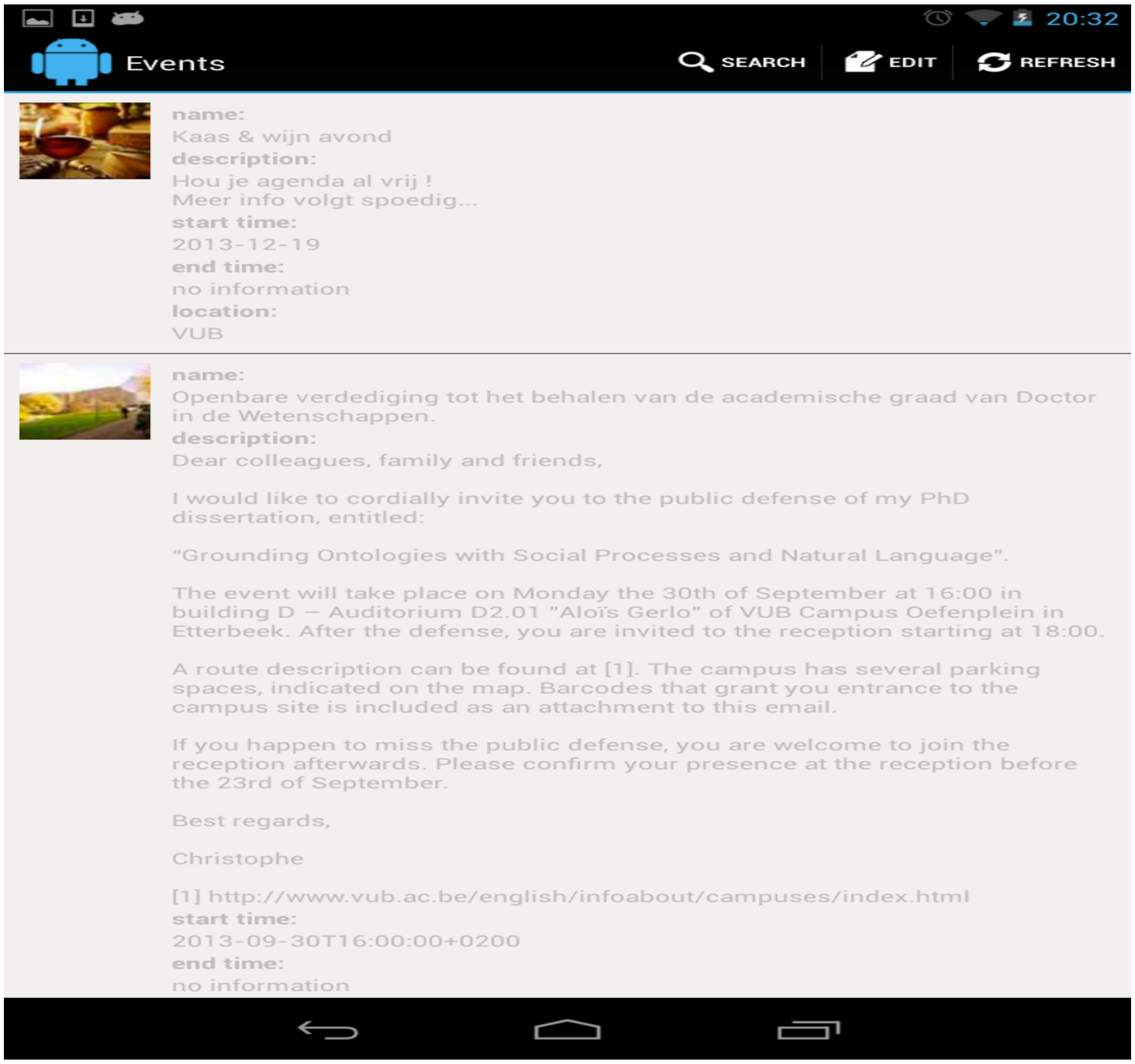
This query is to get all posts that match the user context.

Using Twitter Search API[24] to return a collection of tweets that are relevant to the context. For example, finding all public tweets, which contain the keyword "tcomplex" near VUB area.

https://api.twitter.com/1.1/search/tweets.json/q=%23tcomplex&geocode=50.82469,4.400719,1000&lang=eu&

[24] https://dev.twitter.com/docs/api/1.1/get/search/tweets

5.10 Events Screen



name:
Kaas & wijn avond
description:
Hou je agenda al vrij !
Meer info volgt spoedig...
start time:
2013-12-19
end time:
no information
location:
VUB

name:
Openbare verdediging tot het behalen van de academische graad van Doctor in de Wetenschappen.
description:
Dear colleagues, family and friends,

I would like to cordially invite you to the public defense of my PhD dissertation, entitled:

"Grounding Ontologies with Social Processes and Natural Language".

The event will take place on Monday the 30th of September at 16:00 in building D – Auditorium D2.01 "Aloïs Gerlo" of VUB Campus Oefenplein in Etterbeek. After the defense, you are invited to the reception starting at 18:00.

A route description can be found at [1]. The campus has several parking spaces, indicated on the map. Barcodes that grant you entrance to the campus site is included as an attachment to this email.

If you happen to miss the public defense, you are welcome to join the reception afterwards. Please confirm your presence at the reception before the 23rd of September.

Best regards,

Christophe

[1] http://www.vub.ac.be/english/infoabout/campuses/index.html
start time:
2013-09-30T16:00:00+0200
end time:
no information

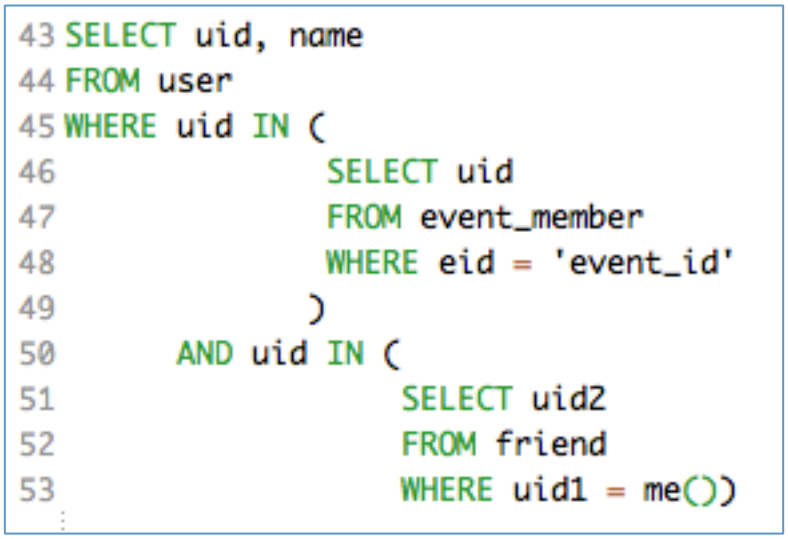
Figure 51 - The Events Screen

The Events Screen is to find all events that belongs to a specific entity, such as building, company, university that are located in the user's surroundings, or events that take place nearby the user's current location. The listing 5.13 illustrates all events that are related to or take place at the VUB University. Below, we explain how we can achieve this result by accessing the Facebook (see section 4.2.2) and Twitter Search API (4.2.3).

```
search?fields=end_time,start_time,location,name&q='keyword_matching'&type=event
```

Figure 52 - Facebook Graph Search to find events related to user's context

This query is to get all events, which match the user context, taking place in the user current neighborhood. This query collects all events related to the user's context. The user's context here is a keyword (keyword_matching) that is passed as a parameter.



```
SELECT uid, name
FROM user
WHERE uid IN (
                SELECT uid
                FROM event_member
                WHERE eid = 'event_id'
               )
        AND uid IN (
                SELECT uid2
                FROM friend
                WHERE uid1 = me())
```

Figure 53 - Facebook Graph Search to find all user's friends who attend a specific event

This query finds out all the user's friends who attend a specific event.

5.11 Search Screen

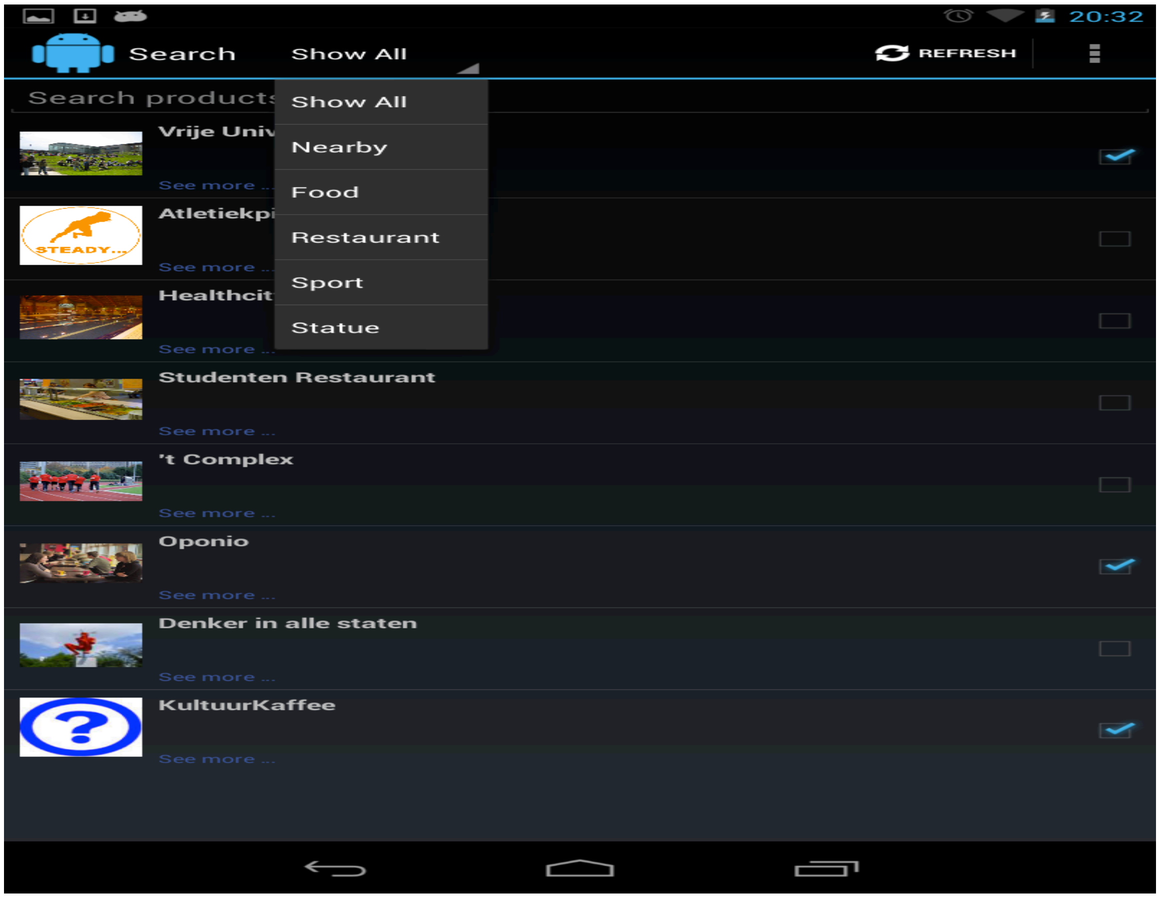


Figure 54 - The Search Screen

The search screen allows user to find context by type, for example nearby entities, detected entities, food shops, restaurants, coffee shops, sport buildings or complex building, statue, and so on. Moreover, the search screen also allows the user to find context by keyword. Tapping on a specific context, he could get more information of this context.

5.12 Map Screen

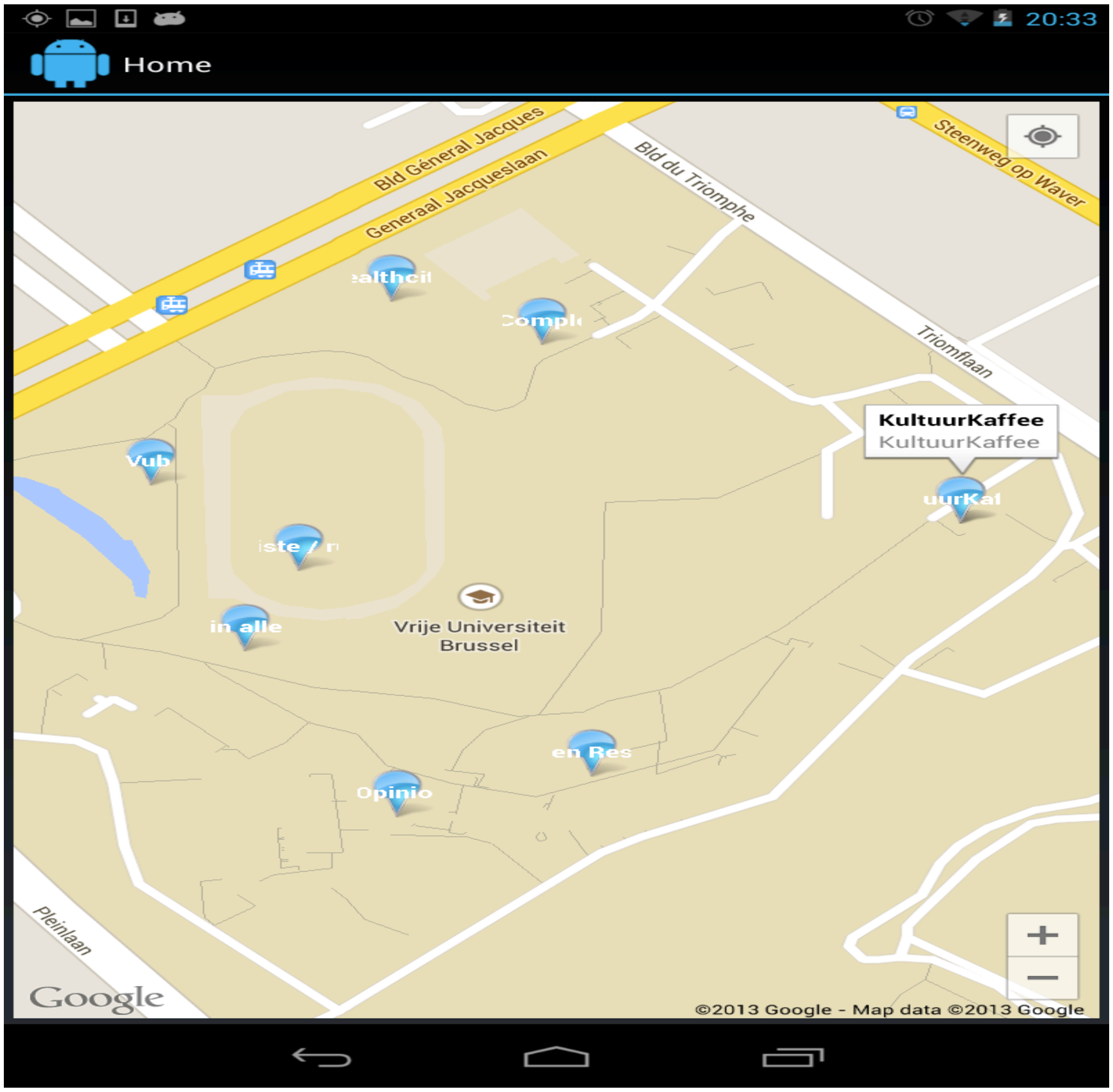
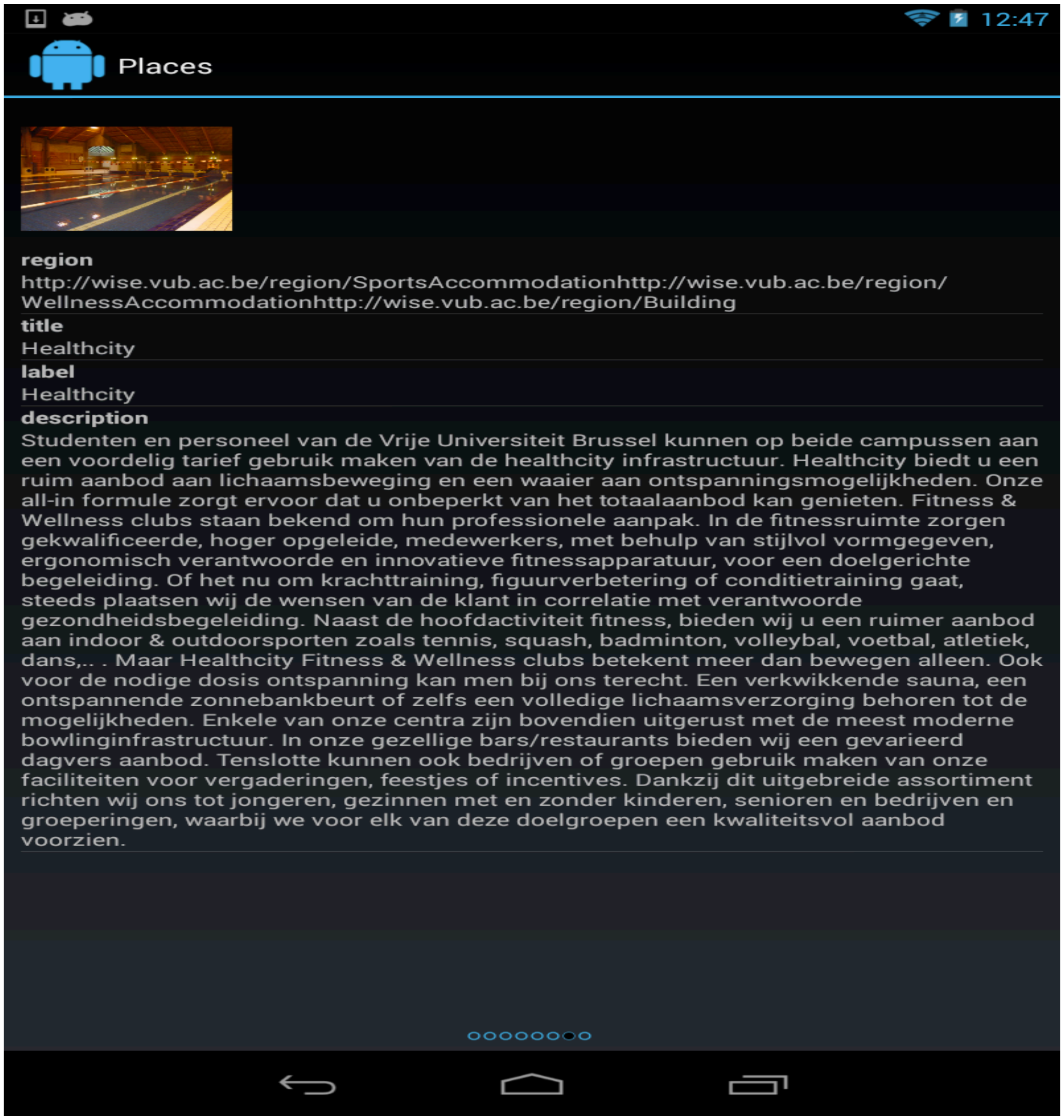


Figure 55 - The Map Screen

Map screen represents all detected entities around the current user's location. The current version of this application doesn't support a feature of showing the path from the user location to a specific entity, but with Google API Service, it can be realized.

5.13 Physical Entities Screen


12:47

Places

region
http://wise.vub.ac.be/region/SportsAccommodationhttp://wise.vub.ac.be/region/WellnessAccommodationhttp://wise.vub.ac.be/region/Building

title
Healthcity

label
Healthcity

description
Studenten en personeel van de Vrije Universiteit Brussel kunnen op beide campussen aan een voordelig tarief gebruik maken van de healthcity infrastructuur. Healthcity biedt u een ruim aanbod aan lichaamsbeweging en een waaier aan ontspanningsmogelijkheden. Onze all-in formule zorgt ervoor dat u onbeperkt van het totaalaanbod kan genieten. Fitness & Wellness clubs staan bekend om hun professionele aanpak. In de fitnessruimte zorgen gekwalificeerde, hoger opgeleide, medewerkers, met behulp van stijlvol vormgegeven, ergonomisch verantwoorde en innovatieve fitnessapparatuur, voor een doelgerichte begeleiding. Of het nu om krachttraining, figuurverbetering of conditietraining gaat, steeds plaatsen wij de wensen van de klant in correlatie met verantwoorde gezondheidsbegeleiding. Naast de hoofdactiviteit fitness, bieden wij u een ruimer aanbod aan indoor & outdoorsporten zoals tennis, squash, badminton, volleybal, voetbal, atletiek, dans,.. . Maar Healthcity Fitness & Wellness clubs betekent meer dan bewegen alleen. Ook voor de nodige dosis ontspanning kan men bij ons terecht. Een verkwikkende sauna, een ontspannende zonnebankbeurt of zelfs een volledige lichaamsverzorging behoren tot de mogelijkheden. Enkele van onze centra zijn bovendien uitgerust met de meest moderne bowlinginfrastructuur. In onze gezellige bars/restaurants bieden wij een gevarieerd dagvers aanbod. Tenslotte kunnen ook bedrijven of groepen gebruik maken van onze faciliteiten voor vergaderingen, feestjes of incentives. Dankzij dit uitgebreide assortiment richten wij ons tot jongeren, gezinnen met en zonder kinderen, senioren en bedrijven en groeperingen, waarbij we voor elk van deze doelgroepen een kwaliteitsvol aanbod voorzien.

Figure 56 - The Physical Entities Screen

The Physical Entities Screen is to represent the information of detected entities in the user's surroundings. More specifically, we obtain this information from the SCOUT framework, and this information is stored in an RDF file on the web. The RDF file contains all information of a specific entity, such as title, region, label, description, image, and so on. Please see section 4.3.1 to see how an entity data type is identified in our system. The Physical Entities Screen provides a list view of all physical entities, and the user can scroll horizontally to see other entities.

5.14 History Screen - The Semi-Auto Posting

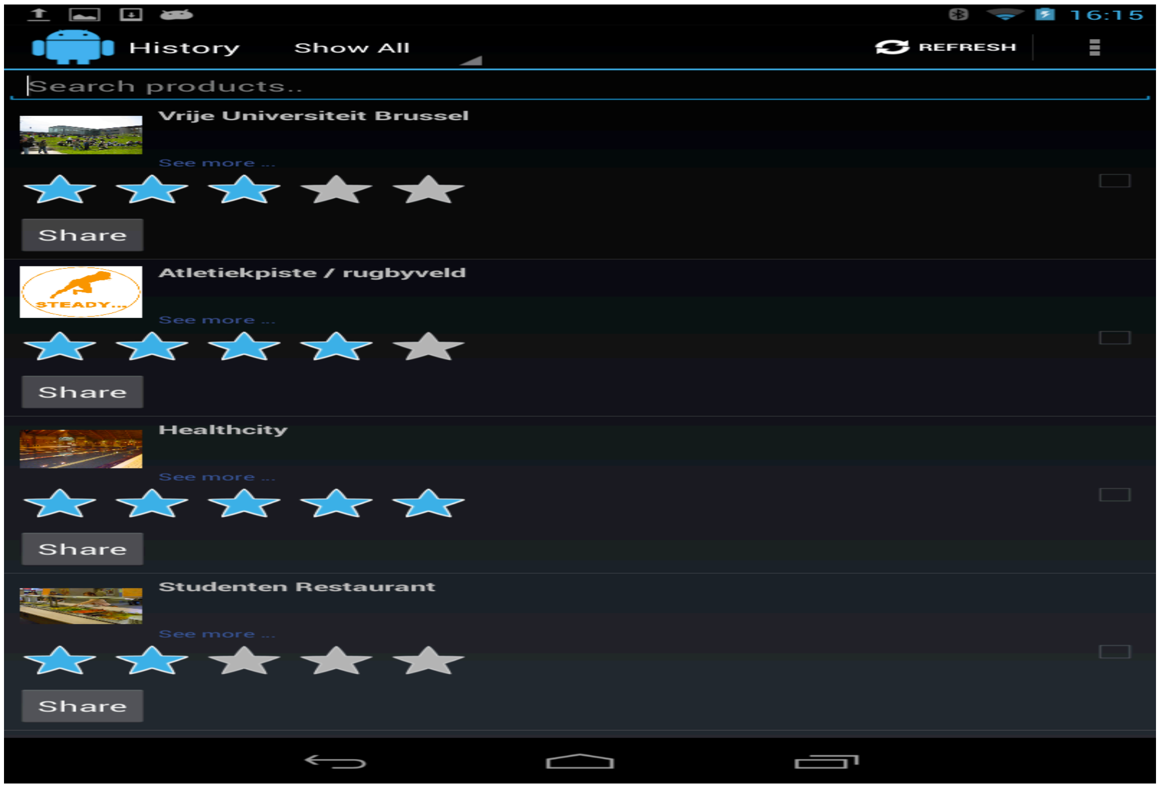


Figure 57 - The History Screen

The history screen shows all contexts that the user has visited during the day. At a specific time of day (e.g., at the end of a day), the user can decide to rate any context he finds interesting, and share it on online communities. This is the semi-auto posting mechanism that we implemented in our application.

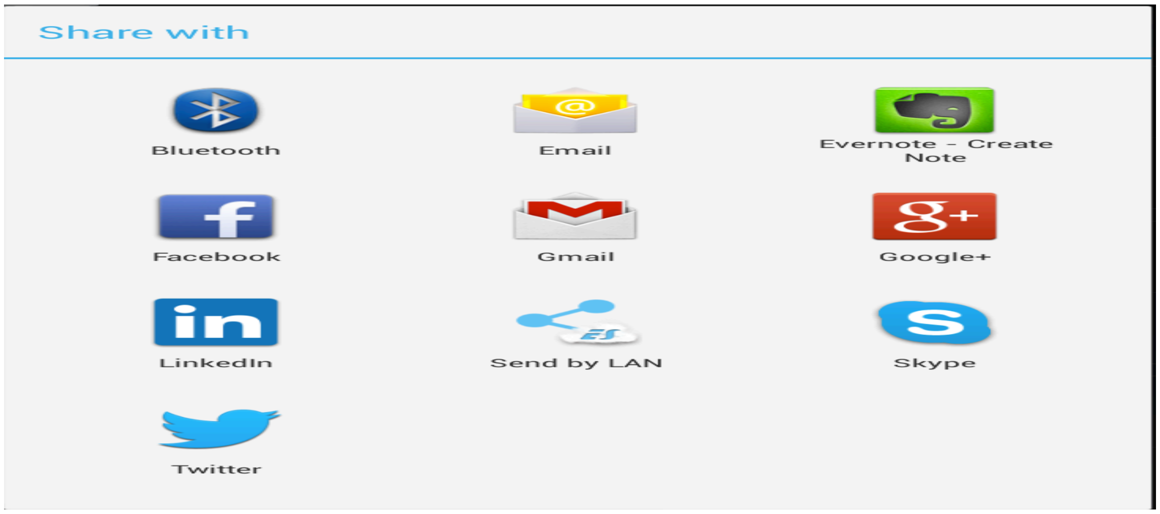
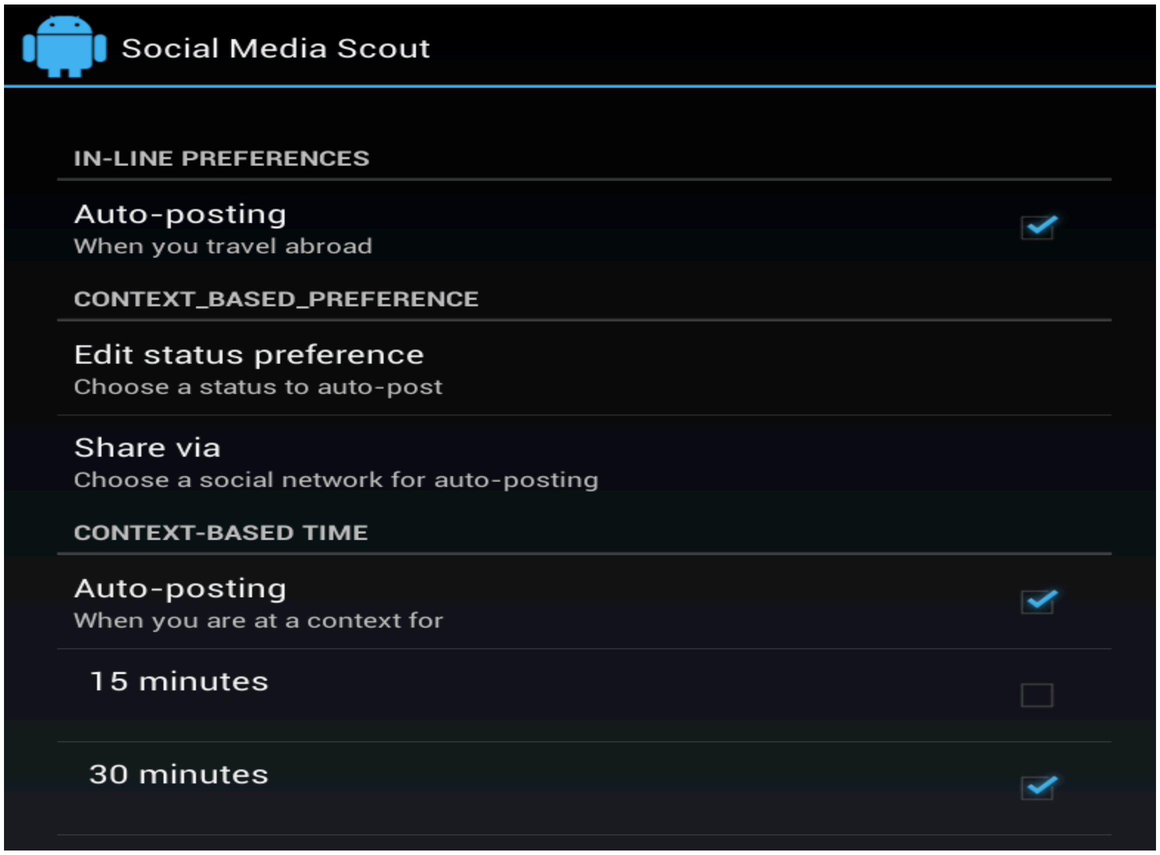


Figure 58 - Share the History Context

5.15 Preference Screen - The Auto-Posting


Social Media Scout

IN-LINE PREFERENCES

Auto-posting ✓
When you travel abroad

CONTEXT_BASED_PREFERENCE

Edit status preference
Choose a status to auto-post

Share via
Choose a social network for auto-posting

CONTEXT-BASED TIME

Auto-posting ✓
When you are at a context for

15 minutes ☐

30 minutes ✓

Figure 59 - The Preference Screen

The Preference screen provides a mechanism to automatically post a status based on the user context, for example in the case you are travelling abroad, or base on the amount of time you are at a specific context (15 minutes, 30 minutes). If you are at a specific context for 30 minutes, maybe you find this context interesting, and the application automatically will post your current location on a chosen social network. You can choose a status and then the location will be added automatically to the post. Below, we explain how we can achieve this auto-posting mechanism.

Firstly, from social networks (e.g., Facebook, Twitter, or Google Plus), we can know the user's home location. Whenever the user moves to another place, our system will determine the user's current location by using the Google Geocoding API[25]. By comparing the user's home location and the user's current location, we can identify if the user is abroad, and the post along with the current location will be shared automatically online.

[25] https://developers.google.com/maps/documentation/geocoding

Secondly, the SCOUT framework provides us a notification service that notifies our application when the user is nearby or far from a specific entity. The Proximity Model in the Environment Layer of SCOUT keeps time-stamped positional relations between the user and physical entities, and thus we can identify how long the user stay at a specific entity. A post, along with the name of entity will be shared automatically online, based on the amount of time that the user stays at a specific entity. This amount of time is set as default in the Preference Screen.

6 Conclusion and Future Work

This thesis presents a generic mobile social media framework. A first contribution of this work is allowing for the integration of any social network. In this vein, a challenging aspect was to resolve the conflicts or overlaps between different data types of the various social networks. We resolved this problem by building a set of shared common data types, enabling us to build a common set of services among multiple social networks.

Moreover, the thesis is also focusing on the challenge of extracting meaningful data based on the user's context from multiple social networks. We proposed a solution of using the context matching function to categorize the context data by degree of relevance, based on the online user profile information, and the user's context information.

A second major contribution lies in achieving a full integration of social media and activities in the physical world. Importantly, this integration is realized in both directions; i.e., from social media to physical world and vice versa. The integration from physical world to social media is illustrated with two mechanisms: auto-posting and semi-auto posting. An auto-posting mechanism is presented, which allows automatically sharing context data in certain situations (e.g., in case the user travels abroad, or stays in the same place for a long time). A semi-auto posting mechanism is also presented, which allows storing all contexts collected during the day; and at the end of each day, the user is able to decide which context should be shared online. On the other hand, the integration from social media to physical world is illustrated by automatically pushing context-relevant social media data towards the user.

In our framework, we decoupled the different concerns into distinct application layers. The Social Network Layer accesses multiple social networks to find relevant information, based on the nearby entities information. The Data Processing Layer determines the degree of relevance of the obtained social network data. The User Interface Layer relies on the returned social media information, and their degree of relevance to automatically generate a UI and display the returned data. The framework is demonstrated by building a mobile application integrating three social networks (Facebook, Twitter, Google Plus) and exploiting six basic kinds of data types (people, photo, check-in, event, group, and post).

Future work could be in the two following aspects:

Firstly, so far, we create a set of common data types in a manual way. By analyzing data types from two or more social networks, the developer should combine and create a unified and common data type in our system. However, building such a common data type is not easy because of the data type structure from different social networks may be complicated, and thus requires a lot of time for the developer to design a common set of data types. Tools or visualization methods could help the developer to understand and analyze different data type structures better, and thereby result into more precise results for combining various data types into a common data type. Such tools would also decrease the developer's efforts.

The integration from physical activities to social media can still be improved. Firstly, user's preferences should be exploited more. For example, frequently visited places can be stored and shared with a group of people. As another example, the time at which the user arrived at work could be shared with his colleagues; while information indicating that he is busy at the moment can be shared with everyone. When the user comes to a favorite place, this information can be shared with his close friends or his girlfriend. In addition, other user context could be collected to provide a better contextual picture. For example, nearby people's information could be extracted from the SCOUT framework, and thus allowing the user to decide to post a status, including nearby people. The user's movements and gestures that are measured by built-in mobile accelerometer can be exploited to allow users to automatically insert their current activity into the status, such as walking, dancing or sleeping.

Bibliography

1. Ahern, S. *et al.* ZoneTag : Designing Context-Aware Mobile Media Capture to Increase Participation. In *Proceedings of the Pervasive Image Capture and Sharing: New Social Practices and Implications for Technology Workshop (PICS 2006) at the Eighth International Conference on Ubiquitous Computing* 3–5 (2006).

2. Beach, A. *et al.* WhozThat? Evolving an Ecosystem for Context-Aware Mobile Social Networks. *IEEE Network: The Magazine of Global Internetworking* **22,** 50–55 (2008).

3. Beach, A. *et al.* Fusing mobile, sensor, and social data to fully enable context-aware computing. In *Proceedings of the Eleventh Workshop on Mobile Computing Systems & Applications - HotMobile '10* 60–65 (ACM Press, 2010).

4. Becker, H. & Gravano, L. Event Identification in Social Media. In *Twelfth International Workshop on the Web and Databases (2009).*

5. Brown, P. J. & Bovey, J. D. Context-aware applications: from the laboratory to the marketplace. *IEEE Personal Communications* **4,** 58–64 (1997).

6. Dey, A. K. Understanding and Using Context. *Personal Ubiquitous Computing* **5,** 4–7 (2001).

7. Dey, A. K. Context-Aware Computing : The CyberDesk Project. In *The AAAI 1998 Spring Symp, on Intelligent Environments* 51–54 (1998).

8. Ferreira, D. R. & Diniz, P. C. Mobile Context Provider for Social Networking. In *OTM '09 Proceedings of the Confederated International Workshops and Posters on On the Move to Meaningful Internet Systems: ADI, CAMS, EI2N, ISDE, IWSSA, MONET, OnToContent, ODIS, ORM, OTM Academy, SWWS, SEMELS, Beyond SAWSDL, and COMBEK 2009* 464 – 473 (Springer-Verlag, 2009).

9. Fujisaka, T., Lee, R. & Sumiya, K. Detection of Unusually Crowded Places through Micro-Blogging Sites. *2010 IEEE 24th International Conference on Advanced Information Networking and Applications Workshops* 467–472 (2010).

10. Gaonkar, S., Li, J., Choudhury, R. R. & Schmidt, A. Micro-Blog : Sharing and Querying Content Through Mobile Phones and Social Participation. In *Proceedings of the 6th international conference on Mobile systems, applications, and services* 174–186 (ACM, 2008).

11. Holleis, P., Wagner, M., Böhm, S. & Koolwaaij, J. Studying Mobile Context-aware Social Services in the Wild. In *NordiCHI '10 Proceedings of the 6th Nordic Conference on Human-Computer Interaction: Extending Boundaries* 207–216 (ACM, 2010).

12. Ji, R., Xie, X., Yao, H. & Ma, W.-Y. mining city landmarks from blogs by graph modeling. In *Proceedings of the 17th ACM international conference on Multimedia* 105–114 (ACM, 2009).

13. Ko, M. N., Cheek, G. P., Shehab, M. & Carolina, N. Social-Networks Connect Services. *The IEEE Computer Society* **43,** 37–43 (2010).

14. Koolwaaij, J. *et al.* Context Watcher ─ Sharing context information in everyday life. In *Proceedings of the IASTED conference on Web Technologies, Applications and Services (WTAS). IASTED* (2006).

15. Labs, A. B. Context-awareness, the missing block of social networking. *International Journal of Computer Science & Applications* **6,** 50–65 (2009).

16. Lu, X., Wang, C., Yang, J., Pang, Y. & Zhang, L. Photo2Trip : Generating Travel Routes from Geo-Tagged Photos for Trip Planning ∗ Categories and Subject Descriptors. in *Proceedings of the international conference on Multimedia* 143–152 (ACM, 2010).

17. Miluzzo, E. *et al.* Sensing Meets Mobile Social Networks : The Design , Implementation and Evaluation of the CenceMe Application. In *Proceedings of the International Conference on Embedded Networked Sensor Systems* 337–350 (ACM, 2008).

18. Pertola, P. & Kolind, L. Context DJ – A Context-Aware adaptive music player. *The Pervasive Computing Course* (2010).

19. Pietiläinen, A., Oliver, E., Lebrun, J. & Varghese, G. MobiClique : Middleware for Mobile Social Networking. In *The 2nd ACM workshop on Online social networks* 49–54 (ACM, 2009).

20. Schilit, B., Adams, N. & Want, R. Context-Aware Computing Application. In *WMCSA '94 Proceedings of the 1994 First Workshop on Mobile Computing Systems and Applications* 85–90 (IEEE Computer Society, 1994).

21. Sigurbjörnsson, B. & van Zwol, R. Flickr tag recommendation based on collective knowledge. In *Proceeding of the 17th international conference on World Wide Web - WWW '08* 327 (ACM Press, 2008).

22. Van Woensel, W., Casteleyn, S., Paret, E. & De Troyer, O. Mobile Querying of Online Semantic Web Data for Context-Aware Applications. *IEEE Internet Computing* **15,** 32–39 (2011).

23. Zanda, A., Eibe, S. & Menasalvas, E. SOMAR: A SOcial Mobile Activity Recommender. *Expert Systems with Applications* **39,** 8423–8429 (2012).

24. Zhao, Q., Liu, T.-Y., Bhowmick, S. S. & Ma, W.-Y. Event detection from evolution of click-through data. In *Proceedings of the 12th ACM SIGKDD international conference on Knowledge discovery and data mining - KDD '06* 484–493 (ACM Press, 2006).

25. Zhu, J., Oliya, M., Pung, H. K. & Wong, W. C. SOLE : Context-Aware Sharing of Living Experience in Mobile Environments. In *MoMM2010 Proceedings of the 8th International Conference on Advances in Mobile Computing and Multimedia* 0–3 (ACM, 2010).