A Middleware for Managing and Sharing Geographical Place Definitions Across Social Networking Services

Jørgen Ekeland
Vegar Engen

Master of Science in Computer Science
Submission date: June 2014
Supervisor: Babak Farshchian, IDI

Norwegian University of Science and Technology
Department of Computer and Information Science

Abstract

This project has been performed as a master thesis and contributes to the UbiCollab project, which is a service-oriented platform for ubiquitous collaboration where social interactions may occur both naturally and unconstrained of situation and location.

Social medias have had a rapid growth throughout the last decade. People tend to share everything they do, and where they do it. With the simultaneously growth of mobile applications, application developers integrate social networking services into their application to reach a greater audience. It is easy to integrate one network into their application, but when multiple networks are integrated, the code tends to become more complex. Developers usually choose only one service to keep it simple. This has the disadvantage of making users of different social medias unable to share their geographical location and communicate with each other.

During our work with this master, we have created a middleware that is able to share and manage geographical places across social networks, in order to make it easier for developers to make location-based applications for multiple social services.

With this implementation we support core features for sharing geographical places in social networks. However, the system needs to support additional features to be treated as a suitable alternative to existing tool.

Keywords: Location sharing, middleware, API, social networking services, geographical places, mobile applications

Sammendrag

Dette prosjektet har blitt utført som et master prosjekt og bidrar til UbiCollab prosjektet, som er en tjenesteorientert platform for allstedeværende sammarbeid, hvor sosiale interaksjoner foregår naturlig og uavhengig av situasjon og lokasjon.

Gjennom det siste tiåret har sosiale medier vokst hyppig. Mennesker liker å dele alt de gjør og hvor de gjør det. Med den parallelle veksten av mobile applikajoner, integrerer utviklere sosiale tjenseter i applikasjonene for å nå et bredere publikum. Det er enkelt å integrere ett nettverk, men når flere nettverk skal integreres i applikasjone, blir koden ofte mer kompleks. Utviklere implementerer ofte bare ett nettverk for å gjøre det enkelt. En ulempe dette medfører er at brukerer av forskjellige sosiale nettverk ikke har mulighet til å kommunisere eller dele geografisk lokasjon med hverandre.

Gjennom vårt arbeid med denne masteroppgaven, har vi lagd mellomprogramvare som gjør det mulig å dele og håndtere geografiske plasser over forskjellige sosiale tjenester.

Med denne implementasjonen støtter vi kjernefunkjonaliteter for deling av geografiske plasser i sosiale nettverk. For å bli et fullverdig alternativ til eksisterende løsninger, må systemet støtte mer funksjonalitet.

Preface

This report documents the work done with UbiNomad, a contribution to the UbiCollab project, written spring 2014 as a Master Thesis. The project assignment was given by IDI (Department of Computer and Information Science, NTNU).

We would like to thank our supervisor Babak A. Farshchian for valuable guidance and feedback during this project. The meetings and conversations have given us motivation and corrections, and have been a good help and support for our final result.

Trondheim, June 11, 2014.

Jørgen Ekeland Vegar Engen

Contents

Abstract **i**

Sammendrag **iii**

Preface **v**

Table of Contents **xi**

List of Figures **xiv**

List of Tables **xv**

1 Introduction **1**

1.1 Motivation 1

1.2 UbiCollab 2

1.3 UbiNomad: A Space Manager for UbiCollab 3

1.4 Project Goals 3

1.5 Deliverables 3

1.6 Research Method and Development Methodology 4

1.7 Report Outline 6

2 Scenarios **9**

2.1 Scenario 1: Where Are My Friends 10

2.2 Scenario 2: The Developer's Struggle 11

2.3 Scenario 3: Sharing Places 12

2.4 Scenario 4: Cultural Differences 13

2.5 Scenario 5: Application Limitations 13
2.6 Summary 14

3 Related technologies 17
3.1 Field of Study 17
3.2 UbiCollab 18
3.2.1 Previous UbiNomad 18
3.2.2 UbiShare 19
3.3 Android Platform 20
3.4 Applications 20
3.4.1 Find My Friends 20
3.4.2 Glympse 21
3.5 Place Providers 22
3.5.1 Google Places API 22
3.5.2 Facebook Places 24
3.5.3 Foursquare 26
3.5.4 Compare Place Providers 28

4 Requirements 29
4.1 Functional Requirements 30
4.2 Non-Functional Requirements 31
4.3 Use Cases 32

5 Architecture 39
5.1 UbiNomad - Architecture 39
5.1.1 General Architecture 40
5.1.2 API 40
5.1.3 External Data Providers 40
5.1.4 Share With Other Applications 42
5.2 Data Model 42
5.2.1 Place Definition 42
5.2.2 Database 44

5.3 Use Cases 45
5.3.1 Create Place 45
5.3.2 Delete Place 47
5.3.3 Edit Place 47
5.3.4 Sync Places 48
5.3.5 Place From External Source 49
5.3.6 Nearby Places 49
5.3.7 Update Current Place 50
5.3.8 Get Current Place 51
5.3.9 Map Places 51

6 Implementation 53
6.1 Database 53
6.2 UbiNomad Content Provider 54
6.3 Database Class Model 55
6.4 Components 56
6.4.1 UbiNomadFragmentActivity 56
6.4.2 PlacePickerList 57
6.4.3 Login fragment 57
6.4.4 Location service 57
6.5 External Place Providers 57
6.5.1 Log in 58
6.5.2 List places near a position 59
6.5.3 Get Single Place 60
6.6 Parse JSON Objects Into Java objects 61

7 Applications 63
7.1 The UbiNomad Manager 63
7.1.1 The Development Process 64
7.2 UbiPost 66
7.2.1 The Development Process 67

7.2.2 Technology 67
7.3 UbiChat 68
7.3.1 The Development Process 68
7.3.2 Technology 69

8 Evaluation **71**
8.1 Evaluation Criteria 71
8.2 Results 72

9 Conclusion **75**
9.1 Discussion and Reflection 75
9.2 Limitations 76
9.2.1 Test With Other Developers 76
9.2.2 Widget 77
9.2.3 Usability Improvement of the UbiNomad Manager 77
9.2.4 Share Aggregator Places 77
9.2.5 Groups and UbiShare 77
9.3 Future Work 77
9.3.1 Update UbiNomad 78
9.3.2 Add Social Networks 78
9.3.3 Automatic Check-in 78
9.3.4 Get Friends' Location 79
9.3.5 Search Places 79
9.3.6 Distribute Library as Binary File 79
9.3.7 Automatically Map Places Together 79

A Android Platform **i**
A.1 Android Architecture i
A.1.1 Linux Kernel i
A.1.2 Libraries ii
A.1.3 Android Runtime ii
A.1.4 Application Framework iii

A.1.5 Applications . . . iii
A.2 Android Activities . . . iii
A.2.1 Fragments . . . iii
A.2.2 Activity lifecycle . . . iii
A.3 Content Provider . . . v
A.4 Sync Adapter . . . v
A.5 Widgets . . . vi

B Getting Started with UbiNomad Library **vii**
B.1 Install Google Play Services . . . vii
B.2 Prepare IDE . . . vii
B.2.1 Errors . . . viii
B.3 Implement In Your Application - Hello UbiNomad . . . viii

C Extend the Library **xi**
C.1 Additional Place Provider . . . xi
C.2 Add Functionality . . . xi
C.3 Example: Foursquare . . . xii

D Terminology **xiii**

E Acronyms **xv**

Bibliography **xvi**

List of Figures

1.1 Time line - Research Method 6

2.1 Where are my friends - Concept 11

2.2 Code bases a developer needs to maintain when using social integration 12

4.1 Use Case Diagram 33

5.1 Sync solution 41

5.3 A place's relationships 44

5.4 Data model 44

5.5 Sequence diagram - Create New Place 46

5.6 Sequence diagram - Delete Place 47

5.7 Sequence diagram - Sync places 48

5.8 Sequence diagram - External place by reference 49

5.9 Sequence diagram - Nearby places 50

5.10 Sequence diagram - Place check in 51

5.11 Sequence diagram - View current place 51

5.12 Sequence diagram - Map corresponding places 52

6.1 Database Schema 54

6.2 Content provider 55

6.3 Database Class Models 56

7.1 UbiNomad Manager Prototype 64

7.2 UbiNomad Manager - Check-in process 65

7.3 UbiNomad Manager - Created Places 66
7.4 UbiNomad Manager - Settings 67
7.5 UbiChat Prototype 68

A.1 Android Architecture [2] ii
A.2 Android Activity Lifecycle[1] iv

List of Tables

2.1 Derivation of Requirements 15

3.1 Compare Providers 28

4.1 Use case 1 - Create place 34
4.2 Use case 2 - Delete place 34
4.3 Use case 3 - Edit place 34
4.4 Use case 4 - View created place 35
4.5 Use case 5 - View external place 35
4.6 Use case 6 - Map places 36
4.7 Use case 7 - Set current place 36
4.8 Use case 8 - Update external place information 37

8.1 Evaluation of the Functional Requirements 72
8.2 Compare criteria pre and post UbiNomad 74

Chapter 1

Introduction

1.1 Motivation

Communicating with friends and colleagues using social networks is becoming a more and more common aspect of people's everyday life. With the growth of social networks, the competition between them is huge and people tend to use the social networks where most of their friends are located. Even though there is a variation of social networks, most developers limit their mobile applications with only one social network integration. It is easier for the developers to only focus on one social network, as long as the chosen social network provides sufficient information to the application. The code written when integrating different social networks differ from network to network. If one developer knows how to integrate one social network, it is not said that he knows how to integrate other social networks. Integration with more that one social network can make the application unnecessary big and complex. With complex code errors, issues and bugs, can easily occur causing the developers to use valuable time debugging the social network implementation. Instead, the programmer wants to use time developing the functionalities in his or her application. Also, if the developer has implemented many applications, he or she has to maintain almost the same code in the different applications.

We want to help the developers solve this time-consuming struggle by making it easier and more efficient to add more than one social network to their application. With just a few method calls and the same method calls with different parameters for the different social networks. Then the developers only need to maintain the code in one place and not for every application.

The focus for our project is location sharing. Therefore we have only looked at integration with social networks that provide the users with locations and places. The terms "location" and "place" are being used interchangeably by most people, which made us create a definition for those terms inspired by the article "Difference Between Location and Place" [5]: A "place" is described as "an area that usually has indefinable margins or borders." In its most generic description, a place is like a segment of space. A place could be for example a school, a city, a market place, a park or a street. A "location" is different from a "place" because you tend to be more specific with the area where a certain spot is situated. Location is something that is described with absolute coordinates (latitudes, longitudes, street numbers, etc.).

Is it possible to make the users share their places with each other across social networks, when different social networks are integrated in the same application? Will this remove some of the limitations regarding who can collaborate together and share places with each other? We want teams that collaborate together, whether it is a student colloquium group or a business team, to easily be able to share locations among them, even if they use different social networks.

1.2 UbiCollab

This project contributes to the UbiCollab project, which is developed by IDI NTNU and SINTEF ICT group for social inclusion projects. As written on the website[9]: "UbiCollab is about supporting natural collaboration using mobile and ubiquitous computing technologies. UbiCollab provides a unique opportunity for you as a student to perform your master's thesis in a vibrant and dynamic research and development environment. [...] UbiCollab is being implemented in Java programming language using state-of-the-art technologies such as Android. UbiCollab is device-centric and most of our development is done on mobile devices.". The main goals for the UbiCollab project are to be able to give students projects with well-defined goals, develop technologies used to support students research in ubiquitous computing, and to encourage students to perform world-class research in the same field. UbiCollab is a platform in constant development.

1.3 UbiNomad: A Space Manager for UbiCollab

This is not the first time someone has written about UbiNomad; another group of students implemented UbiNomad back in 2011 [17]. The term "space" is used instead of our term "place". The previous implementation included a space manager, sample applications and a widget to check-in on a location from the home screen. The space manager made it possible to check-in to predefined spaces, share your check-ins with other applications such as the sample applications, and had the possibility to create, edit and delete your own spaces. You were also able to import and export your spaces in the application. The sample applications that were developed was a calendar, a twitter application, and an application called UbiRule. All of these sample applications used UbiNomad to manage the spaces.

1.4 Project Goals

The goal for this project was to create an API to make it easier for the mobile developers to add more social network integrations into their applications, and to make users from different social networks able to share their location with each other. In this thesis we will use the terms API and library interchangeable. By helping the developers to integrate location based social networks to their mobile applications, with just a few lines of code, the developers will have more time developing and improving the functionalities in their application. This makes it easier for the developers to create location-based applications that are independent of which social network the user is using, and it makes applications available to an increased amount of users.

1.5 Deliverables

This project has been divided into two deliverables. The first delivery was after a prestudy project, written in fall 2013, and the second delivery was after a master project in the spring of 2014. Both projects were related to solving the same issue, however, the prestudy project was mostly focused on doing research regarding the issue. During the prestudy project we investigated the state-of-the-art and other related technologies, created some prototypes and had an architecture discussion. The first delivery is found in the list below:

- A state of the art analysis investigating relevant technologies and related projects
- An architecture document for the API and the data model
- Paper prototypes for a few demo applications and the manager application

The second delivery, which is this thesis, continued where the prestudy project ended. In this project we implemented a solution to the issue based on the architecture discussions from the previous deliverable. This delivery contains a fully implemented API, with a manager application to control the places, and a sample application to test and show the usage of the API. We also delivered an evaluation and conclusion of the solution, and also included some guides on how to install and use the API in an application. The following list shows the second delivery:

- A fully implemented UbiNomad API
- Support of three social networks
- A place manager application
- An architecture discussion
- A sample application that demonstrates the use of UbiNomad APIre
- A discussion and evaluation of the API
- Documentation on how to use the API
- Documentation on how to add an additional place provider
- A getting started guide for the API

1.6 Research Method and Development Methodology

In this Master Thesis we have used scenario-driven development, which mean we based the development on a set of scenarios[14]. This Master Thesis consists of three main phases, as seen in figure 1.1: A research phase, a design and implementation phase, and a validation phase.

Before we wrote this master thesis, we had one semester as a preliminary study where we were using an investigative approach. This early part of the project was called the "Research" phase (seen in figure 1.1). During this phase we did a state-of-the-art analysis and studied the previous UbiNomad implementation, UbiCollab projects and other related technologies (chapter 3). With the gathered information from this prestudy, we created some scenarios in chapter 2 to illustrate some of the problems with location sharing and the previous UbiNomad project. These scenarios were constantly changed during this prestudy. Some were removed and replaced by other scenarios. We did a three step iteration process where we, through searching and reading about related technologies, came up with ideas for some scenarios, created those scenarios, and evaluated them through discussions with our supervisor. After evaluating the scenarios, we adjusted them with new ideas, as shown in figure 1.1. These scenarios played a big part for us during our project and were the foundations for the requirements in chapter 4, and helped shape our final solution. There were also created some use cases(section 4.3) and some paper prototypes during this phase.

The next phase was the "Design and Implementation" phase (figure 1.1). In this phase we developed the API, the manager application and a sample application (see chapter 6 and 7). Before we started the implementation, we designed the data model and the architecture for our solution(chapter 5), and a place definition. The development process was performed using agile software development with sprints lasting between one or two weeks. Working in this environment helped us set concrete goals and gave us rapid feedback on the progress. Agile methodology was a method we both had experience with in previous projects, and with short sprints our supervisor could also track our progress. The three steps in this iterative process was "Idea", "Production" and "Evaluation". In the first step we discussed ideas for our solution, while we designed and developed the solution in the production step, before we had conversations with our supervisor to get feedback and new ideas in the evaluation step. We used those ideas in the next iteration. At the end of the "Design/Implementation" phase we had designed the architecture and the data model and fully developed the library with integration with Facebook and Google Places, a manager application and a sample application.

After the development of the API we entered the "Validation" phase. During this phase we evaluated if our solution supported the requirements in chapter 4. Two of the non-functional requirements were measured to evaluate if they were achieved with our solution.

In the "Design and implementation" phase we included Facebook and Google as place providers in the library. To be able to measure if we had achieved the requirement "UbiNomad should be extendable and modifiable", we waited until the validation phase to implement Foursquare. We managed to implement Foursquare within the expected time frame.

By analyzing the sample applications we developed during the implementation phase, we were able to measure the non-functional requirement number five "UbiNomad should easily be implemented in an application". The amount of code needed to be written, to be able to use the different providers in an application, was less than the set measure, and only one dependency was required to include. In addition to add a few lines of code to the application, an application is required to register at the developer page of each provider. Considering the project setup is not straight forward, it would probably be easier to not use UbiNomad if the developer intends to include only one social network. However, the setup will not be harder, neither will the code be more complex when adding additional social networks, which is the greatest benefit by using our solution.

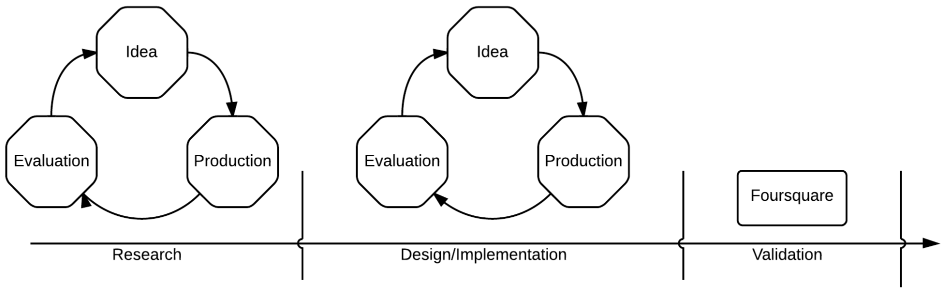


Figure 1.1: Time line - Research Method

1.7 Report Outline

The rest of the report is organized in the following chapters:

Chapter 2 contains a few scenarios that UbiNomad is improving

Chapter 3 presents a state of the art analysis

Chapter 4 presents a set of requirements derived from the scenarios and related technologies

Chapter 5 contains a discussion of the architecture that is going to be used in UbiNomad, including choice of platform and data model

Chapter 6 contains a more detailed aspect of the components in UbiNomad library

Chapter 7 contains a description of applications built on UbiNomad and how they interact with the library

Chapter 8 contains an evaluation of our final results

Chapter 9 concludes the thesis, describes the limitation, and suggests future work that may be done in this scope

Chapter 2

Scenarios

Many applications on the smartphone are dependent upon or take advantage of your GPS location to work properly or give you the information you seek. Some examples are weather forecast, navigation, running and other sports applications, and social network applications. In order to see some of the issues regarding location sharing and collaboration, we outlined a few scenarios describing different situations with location sharing as the main element. These scenarios were written to help us see what expectations a place sharing API should contain, and they were used when creating the requirements for the solution.

It is important to have in mind that when we in these scenarios, and later in this thesis, change between the terms "user" and "developer", we are not referring to the same person. When we use the term developer we refer to the person developing applications who could benefit of using our API. When we use the term user we are referring to the users of the applications the developers have created. It is also important to know that the users of these applications need to have a profile on all the social networks he or she wants to interact with. A user needs a Facebook profile to be able to share a place to Facebook, and so on.

The scenarios presented in this chapter are made up through our own experience, discussions with our supervisor, and by studying related work, including the previous UbiNomad project. Each scenario has its own section, containing a description of the scenario, what the problem is today and how UbiNomad can be of assistance. At the end of this chapter we summarize the problems that we based the requirements upon.

2.1 Scenario 1: Where Are My Friends

This scenario's focus: Allowing users of different social networking services to share places.

John Doe was traveling the country a few years after college, visiting old classmates. Since he was on the road, he thought it would be fun to get in contact with people he had not seen in a very long time. Even though John had been close with a lot of people during his childhood and young adulthood, and friended them on his social networks, he had lost contact with most of them. All together John Doe has 400 friends on Facebook, 30 friends on Foursquare and 100 friends on Google+, scattered all over the world. Of course he can not visit all of them, but if he could figure out who lived close by and see who he could visit during his travels, he would be happy.

By making an application that makes it possible to view a map, and represent each friend with a pin (figure 2.1), he would be able to figure out which of his friends that are nearby the route he wants to travel.

Today there are some applications on the market with this functionality, for example "Find My Friends"[6] which we will discuss further in section 3.4.1. The problem with this, and other related applications, is that the friends you want to follow have to use the same application as you. You can not see other friends' location unless they also use the same application. Another problem is that every developer creating similar applications needs to write almost the exact same code when integrating social networks.

With our solution we can solve both of the issues above. The problem with location sharing in "Find My Friends" is that it only works for users with that application installed on the smartphone. Using our solution the users still needs to have UbiNomad Manager installed on their smartphone, but the location is available for every application that uses the UbiNomad API, not only one specific application. The developers would also be able to get social network integration into their applications with a few lines of code, which is saving the developers a lot of time.

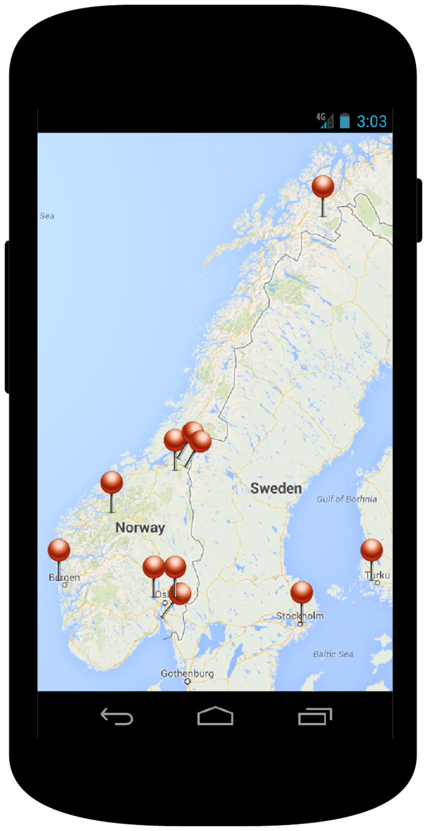
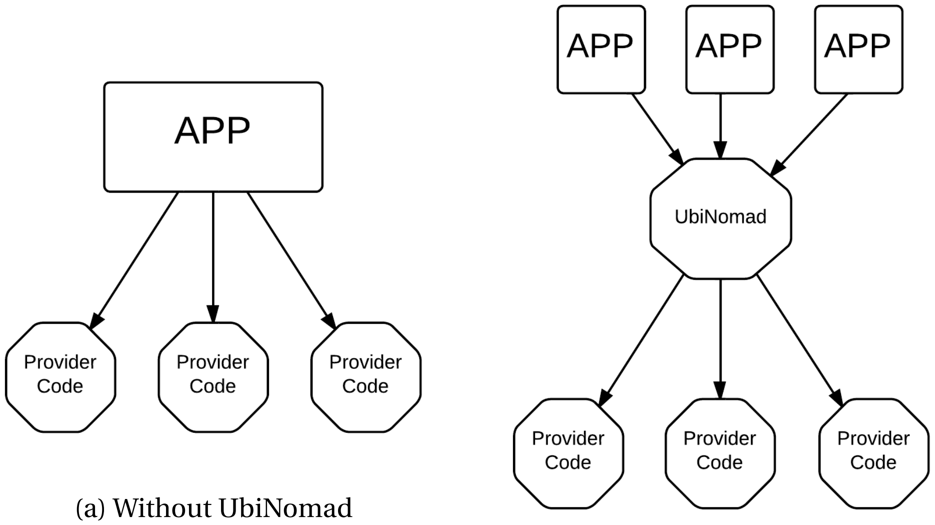


Figure 2.1: Where are my friends - Concept

2.2 Scenario 2: The Developer's Struggle

This scenario's focus: Connecting social integration of different social networks together in one API.

John Doe is an Android developer and has a great idea for a new application that is going to revolutionize the way we cooperate with each other. In the application he wants the users to be able to share their location with the other users. He wants to earn big money on this application and besides developing for different mobile platforms, he wants as many users as possible to be able to use his application, by making social networks users to be able to connect to his application. He is an experienced developer, and this is not his first time developing a smartphone application, but integrating more than one social network to his application requires a lot of time and resources. The time he is using implementing and debugging social integration, is time he wishes that there was an easy way of adding them to his application.



(a) Without UbiNomad

(b) With UbiNomad

Figure 2.2: Code bases a developer needs to maintain when using social integration

The problem with this scenario is that integrating different social networks in an application takes a lot of time and every integration is different for each social network.

Compared to the existing solution for implementing social networks, our solution makes it easier. As seen in figure 2.2, the developer had to manage multiple social networks by himself and use different codes to do the same thing on different networks, while in our solution he may treat all the social networks as one.

2.3 Scenario 3: Sharing Places

This scenario's focus: To be able to make your created place available to other users.

With the previous UbiNomad the users were able to create their own places. John Doe is one of those users and he has made a lot of effort in creating his own places to check-in at. Now Siri, a coworker of John, wants to use the same places as John. The only problem is that John has to export his places, and send them to Siri so that she can import them. John thinks this is unnecessary work and wonders why the locations are not available to everybody when he creates them. This was how sharing your own places was handled with the first implementation of UbiNomad . This is cumbersome and makes the users share their own created places very rarely.

Our solution makes a location available for everyone else who uses UbiNomad immediately after it has been created so that there is no need for exporting places between users.

2.4 Scenario 4: Cultural Differences

This scenario's focus: Allowing users to treat corresponding places from different social networks as one.

One semester John Doe decides to take some classes abroad in China. He quickly realizes that there exists many cultural differences that are absent back home. One of the differences is that the Chinese students are using different social networks than what he is accustomed to. This is because networks like Facebook, Twitter and Google+, that John is using daily, is blocked by the "great firewall of China"[7]. Because they are using different social networks, John thinks of an idea. What if users from different social networks were able to share locations with each other when using different mobile applications.

With UbiNomad, users from different social networks are able to share their locations with each other. They are able to do that because UbiNomad are able to map corresponding places from one social network to a place from another one. If for example The Great Wall of China is a place on Facebook and also in Google Places they are what we call corresponding places. They are two representations of the same physical place. By being able to map places together, applications may treat places from different social networks as one place.

2.5 Scenario 5: Application Limitations

This scenario's focus: Connecting social integration of different social networks together in one API.

Ann is a loyal supporter of Google and uses Google+ as her only social network even though almost all of her other friends are using Facebook. One day a friend of Ann tells her that she needs to install and use the new mobile Application called Tinder [8]. She explains that it is an application to find "hook ups" with boys living nearby. Ann finds this very interesting and downloads the application, but soon she finds out that those applications require a Facebook profile. Ann gets very disappointed that the developers has not made it possible for other social network users to connect to the application and decides to remove the application from her smartphone because it is useless for her.

Some mobile applications require access to the users social network profile to be able to function normally. Tinder is a good example, where the user is not able to use this application unless he or she allows Tinder to access the user's Facebook profile and friendlist. This causes some limitations regarding who can use this application.

There could be several reasons for why the developers choose to only integrate one social network, but this results in the developer choosing the social network with the greatest user database. As stated earlier in scenario "Developers struggle" (section 2.2), we want the requirements for our solution to make it easier for the developers to add social networks to their applications. This will make the applications available for more people and hopefully remove some of the limitations.

2.6 Summary

These scenarios have been created to summarize some of the problems regarding locations sharing we collected from the research phase. People use different social networks because of the following reasons: One reason is which social networks are available and popular in their country, while another reason is which social networks cover their usage and operating environment. A third reason would be who they want to interact with, either by getting in touch with old friends or by meeting new people. Having multiple options of social networks, it exists one that fits the need of one person.

Seen in these scenarios there are some parts that can be improved or where we can make a difference. In table 2.1 we have listed some of the problems regarding these scenarios and what requirement the system shall cover to solve the corresponding problems. These requirements are explained in chapter 4.

Scenario	**Problems**	**Requirements**
1: Find My Friends	• Locations are not available outside the specific application • The developers must write social network integration in every application	• Our solution should be able to share current location with other applications • Our solution shall work with different kinds of location-based applications
2: The Developers Struggle	• Integrating social network requires much time and they are implemented the same way	• Our solution shall work with different kinds of location-based applications
3: Sharing Places	• Cumbersome to share places among users with export and import	• Make places available for all users when they are created
4: Cultural Differences	• Users from different social networks are not able to share locations with each other	• Our solution shall allow users to map places together and treat corresponding places from different social networks as one
5: Application Limitations	• Limitations regarding which social network is integrated in applications • Makes users not able to use some applications because they do not use the required social network	• Our solution should be extendable and reusable

Table 2.1: Derivation of Requirements

Chapter 3

Related technologies

Before embarking on a mission to solve this problem we did a literature review to gain more knowledge. Gaining more knowledge in the field helped us survey the state of the art, such as location sharing, social networks and related applications - both commercial and open source.

Some of the reasons for a literature review:

- give us justifications for our reasons
- figure out if there exists technologies or design patterns that we can take advantage of and use in our project
- get ideas for the implementation of the manager or sample applications
- make sure our solution is compatible with present and future technologies

3.1 Field of Study

Before searching for related work and technologies we had to find our field of study. Our project task aims to create a new generation of UbiNomad where the focus will be on two things: 1)integration with location sharing sites, 2)integration with applications and devices. Because we are implementing a new generation of UbiNomad, it is important for us to read the previous UbiNomad project and get an insight in what they did and achieved.

This thesis is a part of the UbiCollab project, which means we were in need to study a few of the applications previously developed for that group (section 3.2). We also needed to be comfortable with the Android Platform (section 3.3), because one absolute requirement we got with this assignment was that the application had to be written for Android.

We needed to review some of the technologies and applications that already exist with location sharing, and analyze their strengths and weaknesses (section 3.4. We also needed to review some of the most popular social networks with location sharing features and see how they can be used, how the place object is represented and compare their similarities and differences (section 3.5).

3.2 UbiCollab

3.2.1 Previous UbiNomad

The previous UbiNomad project [17] implemented a space manager that made it possible to check into predefined spaces, share the check-ins with other applications such as a calendar and twitter application. The manager also made it possible to create, read, update and delete private spaces. Sharing private spaces with other users required the sender to export his or her spaces, transfer them to the receivers phone, before he or she was able to import them. In order to simplify the check-in process a widget (Appendix A.5) was also developed. The widget contained a list of your spaces, sorted by the most used spaces at the top, and had a one-click check-in system.

The problem this implementation introduced, was that you were only able to make your own places, or places imported from a friend. By handling places in that fashion, UbiNomad acts more like an independent social network than a social network middleware.

Considering the previous UbiNomad implementation acts like an independent social network, the fundamental architecture is built with another goal than what we want to achieve, which was one reason we chose to start over with a new implementation. Nevertheless, we used this implementation and the report as a huge inspiration while we formed our goals and created our implementation. The idea of creating a separate manager, which will be the user interface to the API, is directly linked to this implementation. The user is able to check in and create new places through the manager, which they were able to do in the previous implementation as well, in addition they are also able to map places together.

As the previous UbiNomad was mainly a space manager they needed to define how the spaces looked like, and defined a space as listing 3.1 describes.

```
<Space ID="String">
  <Name>String</Name>
  <Description>String</Description>
  <Parent>
    <Domain URI="String">
      <Id>ID</Id>
    </Domain>
  </Parent>
  <Shareable>Boolean</Shareable>
  <Created>Timestamp</Created>
  <Entities>
    <Entity ID=EntityID>
      <Description>String</Dedcription>
      <Type>EntityType</Type>
      <Data>EntityData</Data>
    </Enitiy>
    ...
  </Entities>
</Space>
```

Listing 3.1: UbiNomad Space Definition

3.2.2 UbiShare

Another component in the UbiCollab project is UbiShare [10]. UbiShare's role in UbiCollab is intended to be the data sharing hub. UbiShare implements a content provider that gives access to a user's social and collaborative data.

UbiShare allows applications to be built on multiple services, which is the same as UbiNomad aims to do. While UbiShare aims to be able to only share files between users, UbiNomad aims for location sharing only. Because UbiNomad and UbiShare both are part of the UbiCollab project, we may be able to use some of the same classes.

For this implementation UbiShare is not that relevant, but for further development it could be used as a great way to distribute AggregatorPlaces between users. It may also be relevant if UbiNomad should support groups, and then make it possible to share files or places through the application.

3.3 Android Platform

As this project is about mobility, it was obvious that we needed to write for a handheld device. Openness is without a doubt an other important aspect of this project, because the UbiCollab project is open-source. All UbiCollab sub-projects are written for android, and considering android developing was a requirement set by our supervisor, the choice of platform was set to Android.

Android is an operating system based on the Linux kernel, primarily built for mobile handheld devices, such as smartphones and tablets. The platform is open source and released under the Apache License by Google [16]. Appendix A will discuss architecture and components in the Android Platform.

3.4 Applications

This section will be discussing a few existing application that uses different location services and the users location, with his or her friends. For each application we will discuss what it does, which problems it solves, and how this application could benefit with an UbiNomad implementation.

3.4.1 Find My Friends

"Find My Friends"[6] is an application created for Android using data from social networks to figure out where the users' friends are located. It is much like the scenario we drafted in section 2.1 which we called "Where Are My Friends". The application marks spots on a map where the user has friends.

The social networks this application is able to get friend information from are Facebook, Google+ and LinkedIn. To be able to get a friend's location, he or she also needs to have a "Find My Friends" account. The implementation is not flexible, because the user has to create an account to be able to use this application.

Using UbiNomad, this application could benefit a lot. The user could be able to find his friends faster, without the hassle to invite them all to use "Find My Friends". The users would no longer need to create an account for this application and all friends, sharing their location, would be viewable for the application. Hence this could also be a disadvantage for established applications, as the openness UbiNomad provides make it possible for new stakeholders to challenge the established stakeholders. All this because a new and an established stakeholder has the same opportunity to get the relevant data.

3.4.2 Glympse

Glympse is a mobile application that gives you the opportunity to share your location with friends over a limited time span. You can also request to make your friends share their location with you. What you do when you want to share your location, is to find who you want to share your location with, and decide for how long you want to share it in real-time with these people. You can also optionally attach a destination place, make Glympse able to estimate time of arrival, and attach a custom message to your "Glympse".

One of the issues with Glympse is regarding privacy. When you start the application all you need to type in is a username. There is no password or other user information required. You have the ability to connect with either Facebook or Twitter, but that is only to save your preferences and history, and is therefore optional. The second issue in Glympse is group sharing. There are two ways a location can be shared in Glympse: either directly to friends by SMS or Email, with a link attached, or you can use the group functionality. These groups are all public for everyone who uses Glympse. It is not possible to make them private. If a user knows the group name, he or she can join the group and access all of the activities in that group. Glympse makes it possible to see all of the members in a group, but that does not make the users feel safe when they share their location. The members listed can still be random people with random usernames. Besides the group sharing, the safest choice would be to share a location directly with friends and not use the groups.

With Glympse, the user searches through his or her contact list on the phone and chooses either a phone number or Email address to send a link to the location. This makes it possible to share your location with users who does not have a smartphone, because they can open the link on a computer.

Some benefits Glympse could make by using UbiNomad, is that it could make it easier to share glympses with friends by using post methods or be able to find more places on the map using place search.

3.5 Place Providers

Throughout this section we will examine a few services that are able to provide us with places. Foursquare and Facebook are in most cases known as location sharing application and could be placed within section 3.4, but they do also make it possible for other applications to use the places they have stored. For each provider we will discuss how they share places, if there are any limitations, and show how a place is represented in that provider.

3.5.1 Google Places API

Google Places API is a service delivered by Google that returns information about places through a HTTP request. A place is defined by Google as either an establishment, a geographical location, or a point of interest. Place requests do specify the location by longitude/latitude coordinates [15]. A place may be returned either as JSON (listing 3.2) or XML. Through the API it is possible to get a list of places, in addition to a more detailed version of a specific location, including user reviews. In addition the API allows us to supplement data to the Google database with our own data from our application. We may add and remove places, schedule events, or even weight rankings from user activity, using Google Bump. Google Bump has replaced the API's check-in functionality, however instead of checking into a place a bump only improves the rank of the place. The Google Places API also gives the opportunity to access millions of photos related to a place.

To be able to use Google Places you need to provide an API key, which can be acquired by getting registered on "https://developers.google.com/". Once an API key is in place, it needs to be included in every request. Places created using an API key, will be available for everyone using the same key, but needs to be reviewed and added to the official Google Places to be globally available.

One thing that limits Google Places is when a place is created, it is not possible to make it private. It will go through a validation process, and may end up as a globally accessible place. In addition Google Places will return a list of places with at most the 20 closest places. Usually this will not be a problem as when a user wants to check in to a place, he or she is probably at, or really close to the place. However, if the user is in a big city with tall buildings, the users' position may only be an approximation, and if there are a lot of "places" nearby, the place the user wants, may not be in the list.

```
{
    "events" : [
      {
        "event_id" : "91J_jK1GfhX",
        ...
      }
    ],
    "formatted_address" : "48 Pirrama Road, Pyrmont NSW, Australia",
    "formatted_phone_number" : "(02) 9374 4000",
    "geometry" : {
       "location" : {
         "lat" : -33.8669710,
         "lng" : 151.1958750
       }
    },
    "icon" : "http://maps.gstatic.com/mapfiles/place_api/icons/
       generic_business-71.png",
    "id" : "4f89212bf76dde31f092cfc14d7506555d85b5c7",
    "international_phone_number" : "+61 2 9374 4000",
    "name" : "Google Sydney",
    "rating" : 4.70,
    "reference" : "CnRsAAAA98C4wD...",
    "reviews" : [
       {
          ...
        },
      ],
    "types" : [ "establishment" ],
    "url" : "http://maps.google.com/maps/place?cid=10281119596374313554",
    "vicinity" : "48 Pirrama Road, Pyrmont",
    "website" : "http://www.google.com.au/"
}
```

Listing 3.2: Example of Google Place API response

Unlike the other social networks in this section, which is both a place provider and a social network, Google Places API is only a place provider. Instead of having a gigantic API, Google has distributed their services into multiple APIs, each with specific tasks. To be able to use the social network Google+, an application has to use the Google+ API.

3.5.2 Facebook Places

In the same way as Google Places API, the Facebook API is able to return places through a HTTP request. A list of places may be returned as JSON after a HTTP call. A place in Facebook contains a longitude and latitude coordinate, a name of the location, a category list, and other attributes shown in listing 3.3.

When using Android we have to use the Facebook SDK for Android. To be able to access places, Facebook requires an authorization token, which we only will get by logging in. Making it possible to log in, the application needs to be registered as a Facebook application in Facebook Developers. When that is done, Facebook GraphPlace are available, which may return a list of places nearby.

Seeing as a user needs to be logged in to access Facebook places, this is a drawback compared to Google Places. Another drawback is that there is a lot of things thats need to be in order just to make it work.

The data model Facebook is using is a Graph database. Facebook has introduced Graph API to make it easier and faster for the developers to access the information they need. This means that when we are searching for a place, Facebook will return a JSON object, as seen in listing 3.3.

```
{
  "about" : String,
  "can_post" : boolean,
  "category" : String,
  "category_list" : [
    {
      "id" : Long,
      "name" : String
    }
  ],
  "checkins" : Long,
  "description" : String,
  "is_published" : boolean,
  "location" : {
    "street" : String,
    "city" : String,
    "state" : String,
    "country" : String,
    "zip" : String,
    "latitude" : Float,
    "longitude" : Float
  },
  "parking" : {
      "street": int,
      ...
    },
    "phone" : String,
    "talking_about_count" : Long,
  "username" : String,
  "website" : URL,
  "were_here_count" : Long ,
  "id" : Long,
  "name" : String,
  "link" : URL,
  "likes" : Long,
  "cover" : {
    "cover_id" : Long,
    ...
  }
}
```

Listing 3.3: Facebook Place definition

3.5.3 Foursquare

Foursquare is a location-based social network that aims to make cities more interesting to explore. Foursquare is used to find friends, as a social city guide, and as a game that challenges users to experience new things. The users get points when they check-in at a place, which is done either through a mobile website, a text message, or a smartphone application. Besides the game, the check-ins are also used to tell friends where they are and with whom, and track the history of where they have been.

The Foursquare API also returns information about places through HTTP request and works pretty much the same way as Google Places API, return venues as JSON (listing 3.4).

```
{
  id: "40b28c80f964a520affb1ee3",
  name: "House of Blues",
  contact: {
    phone: "+13129232000",
    ...
  },
  location: {
    address: "329 N Dearborn St",
    crossStreet: "btwn Kinzie St & Wacker Dr",
    lat: 41.888153911029704,
    lng: -87.62902319413992,
    postalCode: "60654",
    cc: "US",
    city: "Chicago",
    state: "IL",
    country: "United States"
  },
  categories: [
    {
      id: "4bf58dd8d48988d1e5931735",
      name: "Music Venue",
      ...
    }
  ],
  verified: true,
  stats: {
    checkinsCount: 26524,
    ...
  },
  url: "http://www.houseofblues.com/chicago",
  likes:
  {
    count: 933,
    ...
  },
  like: false,
  rating: 9.57,
  reservations: {
    url: "http://www.opentable.com/single.aspx?rid=39712&ref=9601"
  },
  hours: {
    isOpen: false
  },
  photos: {
    count: 1358,
  },
  hereNow: {
    count: 0,
  }
}
```

Listing 3.4: Example of Foursquare Venue Response

3.5.4 Compare Place Providers

As seen in table 3.1 all the three providers we have compared supports mostly the same features, which makes a lot of sense, because the features they support are basic features for a map service. The only way they differ is how easy it is to add a new place and if it is available whether or not the user is logged in. Based on those observations it makes sense to implement all three of these providers, but make Google Places our main provider, as it is the only one available without the need of login.

Even though each of the providers define places a bit different, they all return places as JSON objects which make it easy to transform them to a model of our choice. However, by using the most common definitions for each field, we created our own place definition (section 5.2.1).

	Google Places	**Facebook**	**Foursquare**
List places near a location	✓	✓	✓
Get place by id	✓	✓	✓
Place have category	✓	✓	✓
Place includes location	✓	✓	✓
Returns place as JSON	✓	✓	✓
Returns place as Object	✗	✓	✗
Add place through HTTP	✓	✗	✓
Available without login	✓	✗	✗

Table 3.1: Compare Providers

Chapter 4

Requirements

The requirements of a software system are a set of prerequisites for the system and used as guidelines for what the system shall or shall not do. Setting up requirements is one of the most important tasks during software development and it is usually done early in the development process to set the frames and behavior for the system.

The requirements for our solution were set during the research phase. They were gathered from the previous UbiNomad project, through the scenarios, in chapter 2, and some were added and removed through conversations with our supervisor. The derivation of the requirements from the scenarios can be seen in table 2.1. During the design and implementation phase we used these requirements as guidelines for what functionalities and behavior the library and the manager should contain. The requirements were divided into two parts: functional requirements and non-functional requirements. The functional requirements were mostly related to the user's interaction with the manager application, and the non-functional requirements were related to the behavior of the library.

The requirements were also used in the evaluation and conclusion of our solution in chapter 8. Evaluating both the functional requirements and the non-functional requirements is usually a different task. The functional requirements are mostly measured in the implementation's supported functionalities, and the non-functional requirements are usually measured with some criteria for how the solution shall behave in certain situations.

4.1 Functional Requirements

The functional requirements for our solution are presented in the list below:

1. UbiNomad shall allow users to create places
2. UbiNomad shall allow users to edit places
3. UbiNomad shall allow users to delete places
4. UbiNomad should be able to get places from external sources
5. UbiNomad shall allow users to view their places
6. UbiNomad shall allow users to update current place from external sources and created places
7. UbiNomad shall allow users to map places together
8. UbiNomad should share created places between the users
9. UbiNomad should be up to date with places from external sources
10. UbiNomad should be able to display the right information about a place to the user
11. UbiNomad should be able to share current location with other applications

The first three requirements let the user create, edit, and delete places. These requirements were taken from the previous UbiNomad implementation and are essential for the user to represent his or her environment of places. With this requirements our solution lets the user manage places without connecting to a location based social network.

The fourth requirement makes the user able to get places from external sources like Facebook, Foursquare and Google Places in addition to the created places covered in requirement 1.

The fifth requirement allows the user to view his or her nearby places and information about them. This applies to both the created places and the external places.

Requirement six is an action related to the places. With this requirement the users are able to check-in or update their current location, both with the created places and the external places. Without this requirement the user would not be able to use the places.

The seventh requirement allows the user to map places with each other whether it is a created place or an external place. Giving the user this ability we cover the problem regarding users using different social networks. Now the users are able to share their locations across different social networks.

Requirement eight states that a created place shall be shared and available for all the users. As soon as a place is created it will be available in the network for all the other users.

The ninth and tenth requirement are created so the user can view the right information about the places. If an external place is changed, the changed information shall also be changed in our solution. This way the external places have always the correct information.

The last requirement is created so that applications, that uses our solution, shall be able to get the user's place. This requirement is very important for our solution and it would be useless without this requirement. If the applications could not get the user's current place from our API, there would no point in using our solution either.

4.2 Non-Functional Requirements

While functional requirements describe what the system should do, non-functional requirements describe how the system will behave. The non-functional requirements are constrains that describe what the system should be. These are the non-functional requirements for our solution:

1. UbiNomad should be developed using only open source technology
2. UbiNomad should be developed for Android
3. UbiNomad should be extendable and modifiable
4. UbiNomad shall work with different kinds of location-based applications
5. UbiNomad should easily be implemented in an application

6. The place-definition should be extendable

7. The place-definition should be compact

The first two non-functional requirements were determined by UbiCollab. UbiNomad has to be kept as an open source project and all software are written for Android.

The social networks we integrate in our solution, are in constant change. Big changes can cause our solution to fail accessing the external places from the providers. We added the third requirement to be able to modify our solution when changes occur. We also require our API to be extendable, so the developers can add new features to our solution and add more social network integrations.

The fourth requirement states that our solution shall work with different kinds of location-based applications. Our API shall be usable for different type of mobile applications, which are dependent on the user's location.

One of the goals for our solution is to provide help for the developers when creating location-based applications. If our solution is difficult to integrate in the developer's application, our goal is not achieved, hence we added the fifth requirement.

The sixth non-functional requirement is important because a Google Place look different than a Foursquare Venue, and other fields may be appropriate. Because of the differences in the place-definition between the different place providers, it must be extendable.

The seventh and last non-functional requirement is added because we do not want to store more information than necessary, but at the same time make the applications find relevant information.

4.3 Use Cases

A use case describes the interaction between actors and a system. A use case is usually a step-by-step description of the flow of events happening after an actor uses the system to achieve a specific goal. In our use cases (figure 4.1) we have two actors, a user and an application. The application actor is the applications that uses our API in the development, and the user actor is the one using our developed manager application (described in section 7.1).

We have also created some textual use cases, to the corresponding use cases in figure 4.1, to describe in more detail how the interactions are performed with the manager application. These textual use cases are limited to only cover the interaction of the user actor. For more detail in how the interactions is done in with the Ubinomad Manager, see section 7.1, and for a more detailed use case regarding the library and the database see section 5.3.

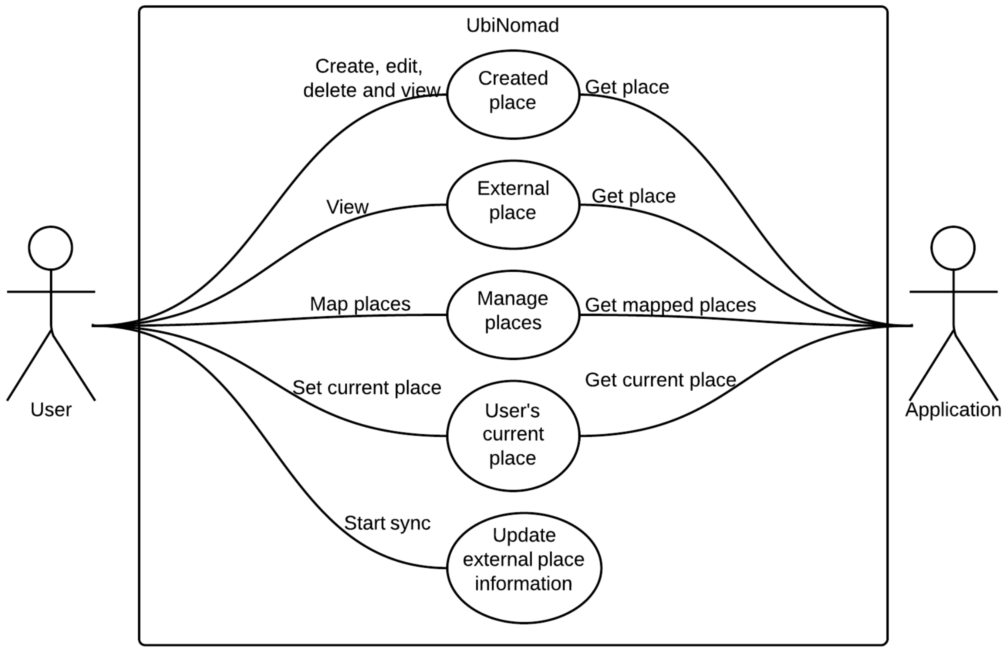


Figure 4.1: Use Case Diagram

The expressions used in the textual use cases, and their meaning are listed here:

Precondition Describes the state of the system before the scenario has begun

Postcondition Describes the state of the system after the scenario has been performed

Success scenario Describes the events of actions that makes the scenario a success

Extension Describes the performed actions performed when the scenario is not a success

Use case ID	1
Name	Create place
Precondition	Application is running and "My places" tab is pressed
Postcondition	The created place has been added to the database
Success scenario	Step 1: Press the "Create place" button Step 2: Fill in name, category and other necessary information Step 3: Press the save button Step 4: The system adds the place to the database
Extension	If something goes wrong, all temporary changes shall be undone

Table 4.1: Use case 1 - Create place

Use case ID	2
Name	Delete place
Precondition	Application is running and there exists a place the user has created and "My places" tab is pressed
Postcondition	The place has been removed from the database
Success scenario	Step 1: Long press the place you shall delete Step 2: The system displays a "Delete button" Step 3: Press the "Delete button" Step 4: The system removes the place from the database
Extension	If something goes wrong, all temporary changes shall be undone

Table 4.2: Use case 2 - Delete place

Use case ID	3
Name	Edit place
Precondition	Application is running, there exists a place the user has created and "My places" tab is pressed
Postcondition	The place has been updated with the new information
Success scenario	Step 1: Long press the place you shall delete Step 2: The system displays a "Delete button" Step 3: Press the "Delete button" Step 4: The system removes the place from the database Step 5: Press the "Create place" button Step 6: Fill in name, category and other necessary information Step 7: Press the save button Step 8: The system adds the place to the database
Extension	If something goes wrong, all temporary changes shall be undone

Table 4.3: Use case 3 - Edit place

Use case ID	4
Name	View created place
Precondition	The application is running and the user has created a place
Postcondition	The user views a list of created places
Success scenario	Alternative 1: Step 1: Press the "My places" tab Step 2: The system displays a list of the user's created places with relevant information Alternative 2: Step 1: Press the "Nearby places" tab Step 2: The system displays a list of nearby places sorted on distance. If the created place is nearby it will be displayed in this list with relevant information
Extension	Alternative 1: List will be empty if there exists no created places owned by the user Alternative 2: List will not contain the place if the user has not created any place or the created place is not nearby

Table 4.4: Use case 4 - View created place

Use case ID	5
Name	View external place
Precondition	The application is running
Postcondition	The user sees the places from the external sources
Success scenario	Step 1: Login to the social networks the user wants places from in the "Settings" tab. By default, only "Google Places" provides external places to the user Step 2: Press the "Nearby places" tab Step 3: The system displays a list of nearby places gathered from the connected social networks, sorted on distance
Extension	

Table 4.5: Use case 5 - View external place

Use case ID	6
Name	Map places
Precondition	The application is running and there exists a place in the list of nearby places
Postcondition	The place is added to an aggregator place
Success scenario	Step 1: Press the "Nearby places" tab Step 2: The system generates a list of places nearby your location Step 3: Press the place from the list the user want to add to an aggregator place Step 4: The system displays a list of aggregator places which includes the previous selected place Step 5: The user press the "Add new Aggregator place" button Step 6: The system displays all the users aggregator places Step 7: The user press the aggregator place the user wants to add the place to Step 8: The system adds the place to the pressed aggregator place
Extension	If the manager list of aggregator places is empty or does not contain the right aggregator place in step 6, the user press "Create aggregator place". The system creates a new aggregator place and and adds the pressed place to the created aggregator place

Table 4.6: Use case 6 - Map places

Use case ID	7
Name	Set current place
Precondition	The application is running and there exists a place in the list of nearby places
Postcondition	Current place is set for the user
Success scenario	Step 1: Press the "Nearby places" tab Step 2: The system generates a list of places nearby your location Step 3: Press the place from the list the user want to set as current place Step 4: The system displays a list of aggregator places which includes the previous selected place Step 5: The user press the prefered aggregator place and the system updates the user's current location
Extension	If the prefered aggregator place, in step 4, is not displayed in the list, the user press "Create new Aggregator Place". The system then adds the selected place to the aggregator place and the system updates the user's current location

Table 4.7: Use case 7 - Set current place

Use case ID	8
Name	Update external place information
Precondition	The application is running
Postcondition	Place information is updated
Success scenario	Step 1: Press the "Sync" button Step 2: The system updates the information for all the places saved in the database
Extension	

Table 4.8: Use case 8 - Update external place information

Chapter 5

Architecture

To be able to reach the goal for this project, which is to make it easier to develop location based applications for multiple social networks, we needed to create a tool. We called this tool UbiNomad and this chapter will focus on how we built the UbiNomad API. This includes the choice of architecture we are using, which platform it is build for, and other technologies we make use of.

This chapter will be divided into several sections. First, in section 5.1 we will discuss the architecture for UbiNomad API, including how external providers are used, what the API supports, how we are handling creation of new places, and how to share data with other applications.

In section 5.2 we will discuss the data model we are using to represent places in UbiNomad. While, in section 5.3, we will present how our previous defined use cases are implemented in the library.

5.1 UbiNomad - Architecture

This section will take a closer look at how we have built the UbiNomad Library. In section 5.1.1 will we discuss the general architecture of the application, before we go more in detail of what the API supports (section 5.1.2), and how we are handling external data providers (section 5.1.3). In section 5.3.1 we will discuss different approaches to creating a new place, before we end it in section 5.1.4 with how places are shared with other applications.

5.1.1 General Architecture

As a requirement for UbiCollab projects is to develop for Android devices, we had no choice in the matter, and designed this library for Android. The architecture of the UbiNomad API, as seen in figure 5.1, consists of one sync adapter for each external provider, an API that can be used by different applications, and grant access to a content provider. UbiNomad also supports the possibility of adding places from external sources, like Facebook or Google.

Android has a built-in SQLite database which we have taken advantage of. When something is requested through the API, the library will get it from the internal database, if - and only if - the record does not exist it will get it from the appropriate data provider instead.

UbiNomad also comes with an API which gives third-party applications the opportunity to get user data, groups and places. The API will be fully discussed in section 5.1.2

In addition to an API, UbiNomad is delivered with a manager, which will make it possible to view all nearby places, check in to a nearby place, and add or delete your own places.

5.1.2 API

The API delivered with UbiNomad is a CRUD API. It makes it possible to create, read, update and delete places, check in to a place, and create a status update to each provider containing the location linked with that provider, through the API.

Before an application is able to use the UbiNomad API, the UbiNomad library has to be included in the project. The library makes it possible to integrate an application in multiple social networks, and supports methods for login and status updates.

5.1.3 External Data Providers

The possibility to retrieve data from external sources raised a few questions: What do we want to import? When do we import data? How much data do we want? How should it be imported? How would the data be kept up-to-date? This section will describe our thoughts on those questions.

There is a significant amount of place data available in all of the providers we encountered. Obviously it would not make any sense to store a local copy of every place as it would only waste and exceed the storage capacity of the device. We figured the places we want to store, are the places the user is actually using, which brings us to our second question.

We did establish that we only need to store places the user is using. The most logical time to store them would be at check-in, because the user is not using other places than the places he is checking in to.

The amount of data we wanted for each place was determined by looking at the place definition (section 5.2.1).We did not want to store any data beyond what we already had said we would need.

To be able to keep the data up-to-date we needed to compare the stored raw place to the original place. Since the application framework already has sync adapters (section A.4) implemented, we figured we could make use of them. One of the positive consequences of using a sync adapter is that the places may be synced automatically, for example at set times. As figure 5.1 shows, we needed a sync adapter for each provider. This makes the library a bit harder to extend, but keeps the architecture clean.

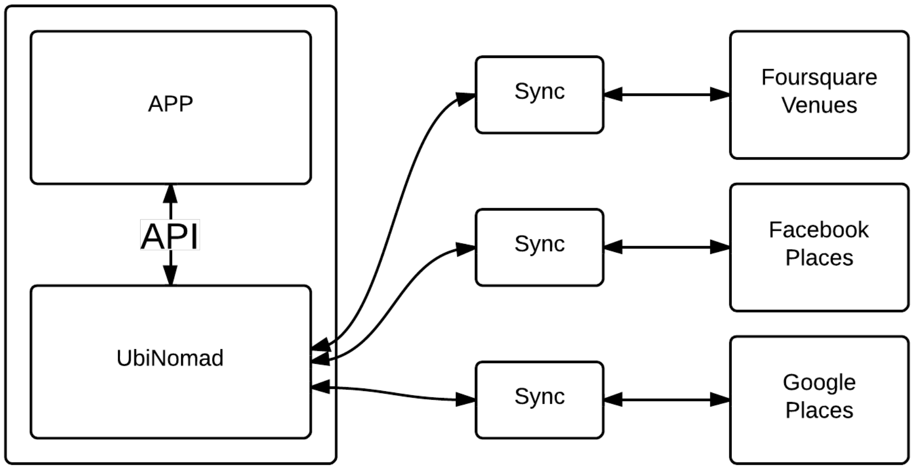


Figure 5.1: Sync solution

By using this approach we get a nice side effect. Seeing that all the providers are able to list nearby places, we do not to use the devices computation power to find the nearby places stored on the device.

5.1.4 Share With Other Applications

Based on requirement 1 "UbiNomad should be able to share current location with other applications" we had to find a way to fulfill the requirement. As all data used by the library is stored in a SQLite database, accessible through a content provider. The content provider encapsulates the database, as described in section A.3. By using this interface it is possible for other applications to get access to data stored by the library, this includes the current location.

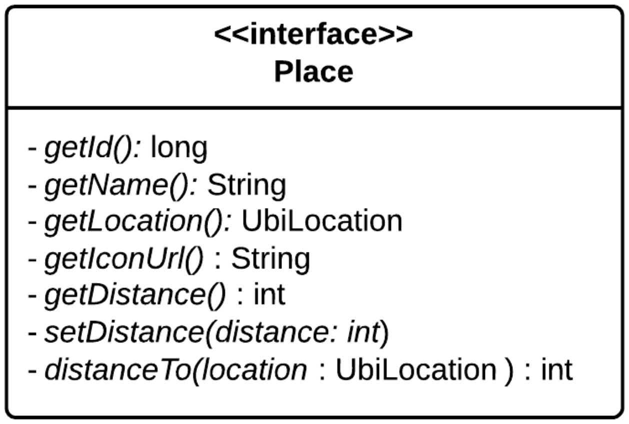
5.2 Data Model

5.2.1 Place Definition

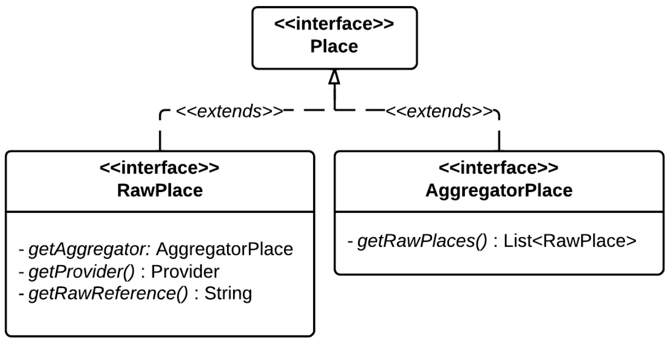
In chapter 3 we described a few APIs regarding places. They all responded to a HTTP request and returned JSON definition of the place. They were all quite similar, but also had a few differences. All the returned places had an attribute "location" which contained another object with latitude and longitude, in addition Foursquare and Facebook also returned an address in the location object. Google Places, on the other hand, had defined the address attribute in the place object, not location object. For the most part they agree on which attributes they use, but they do not always agree on what to name them.

To make the place definition easy to extend and modify we are suggesting defining an easy interface, as shown in figure 5.2a. Using an interface makes it possible to use multiple implementations of the place.

The seventh functional requirement from chapter 4 is "UbiNomad shall allow users to map places together". By adding two fields to the place, provider and providerReference, a place could be mapped towards an external source. We also needed to gather corresponding places together. One place may have zero, one or many corresponding places, which may look like a one-to-many relationship. However, because a Place has a one-to-many relationship with itself, this will be handled as many-to-many relationship. To be able to keep track of this many-to-many relationship, we added a new object called AggregatorPlace. The AggregatorPlace has a one-to-many relationship with the external places, which will be call RawPlaces from now, as seen in figure 5.3.


<<interface>>
Place
- getId(): long
- getName(): String
- getLocation(): UbiLocation
- getIconUrl() : String
- getDistance() : int
- setDistance(distance: int)
- distanceTo(location : UbiLocation) : int

(a) Place Definition


<<interface>>
Place
<<extends>>
<<extends>>
<<interface>>
RawPlace
- getAggregator: AggregatorPlace
- getProvider() : Provider
- getRawReference() : String
<<interface>>
AggregatorPlace
- getRawPlaces() : List<RawPlace>

(b) Place Definition with relations

Based on the fact that the aggregator only exists on a device, the external source references would be excess data. This left us with two different types of places: Places gathered from an external source, which we called raw places, and places representing a set of corresponding raw places, which we called aggregator places. Both of these place types extend Place, shown in figure 5.2b

While gathering external places we will transform a external place into a RawPlace by using a wrapper to return the appropriate values from the external place.

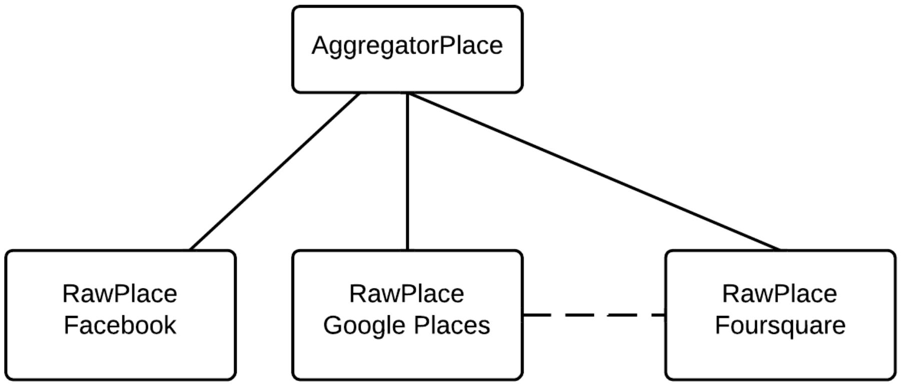


Figure 5.3: A place's relationships

5.2.2 Database

UbiNomad is using a SQLtite database, because it is natively supported in Android, and because a database is less time-consuming than working with text files. The data structure is modeled as an ER-diagram (figure 5.4). As seen in the diagram, a user has a current location which is an AggregatorPlace. When a place is imported from an external source it is saved as a raw place marked with the provider it was imported from (Google, Facebook, Foursquare).

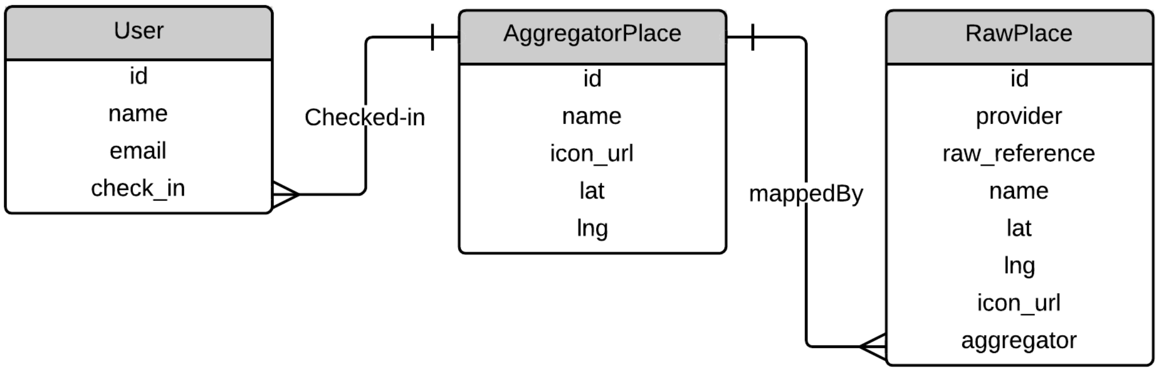


Figure 5.4: Data model

5.3 Use Cases

In this section we will explain in more detail the use cases from section 4.3. However, in this section we are most interested in what is happening in the library and what method calls are executed, therefore we will not be presenting where it will be found in the manager in this section. How the user interacts with the UbiNomad Manager are explained with pictures in section 7.1.

The functionality of the library will be presented on an abstract level. We will only present how the different components of the library are interacting, and a more detailed view of components will be presented later in chapter 6

5.3.1 Create Place

One of the requirements tells us that the user should be able to add and manage their own places. This leaves us with a few different solutions, and throughout this section we will discuss the alternatives.

One possible solution is to add the place to an external provider. Choosing that solution gives us the the easy way out, because we do not need to store anything of our own. The disadvantages with this solution is that we do not have the power to do whatever we want with the place. The place has to follow the rules of the providers, which means that we lose the the possibility of having private places.

Another possibility is to only store the place locally on the device. By doing this we have secured the place privacy issue, and we have full control over the place. We are still not in need of a server in this solution either. Sharing a place with someone needs to be done by exporting the place to another medium, before importing it to another device which is a huge drawback with this solution.

The third solution uses a server to store all our places. It could be used in the same manner as Google; a simple REST service. We will send a request for places and a JSON object will be returned. The advantage of using this solution is that we will be in full control of whether or not a place is public, or shared with only a few people. It will also give us the opportunity to easily share a place with other users. In this case we will need to have a running server, but making the server open source would not only lessen the capacity we need to support, but it will also strengthen the privacy. Since you can run a private server, you will know who has access and be able to eliminate an unwanted third-party. There are however a few disadvantages choosing this approach. It is harder to set up, because we need to have a server running a place provider service. If we should make the server code open-source we would also need to make it possible to change which server the applications should be using. Also, a server set up by us has a higher probability of being unavailable due to technical difficulties or blackouts.

Based on the advantages and disadvantages for each approach we decided to use the first alternative, by adding the place to Google Places. The reason behind it is that it was a really simple solution. The created place will look just like another Google place, while the creator has the possibility to delete it. The sequence of what will happen when creating a new place is described in figure 5.5.

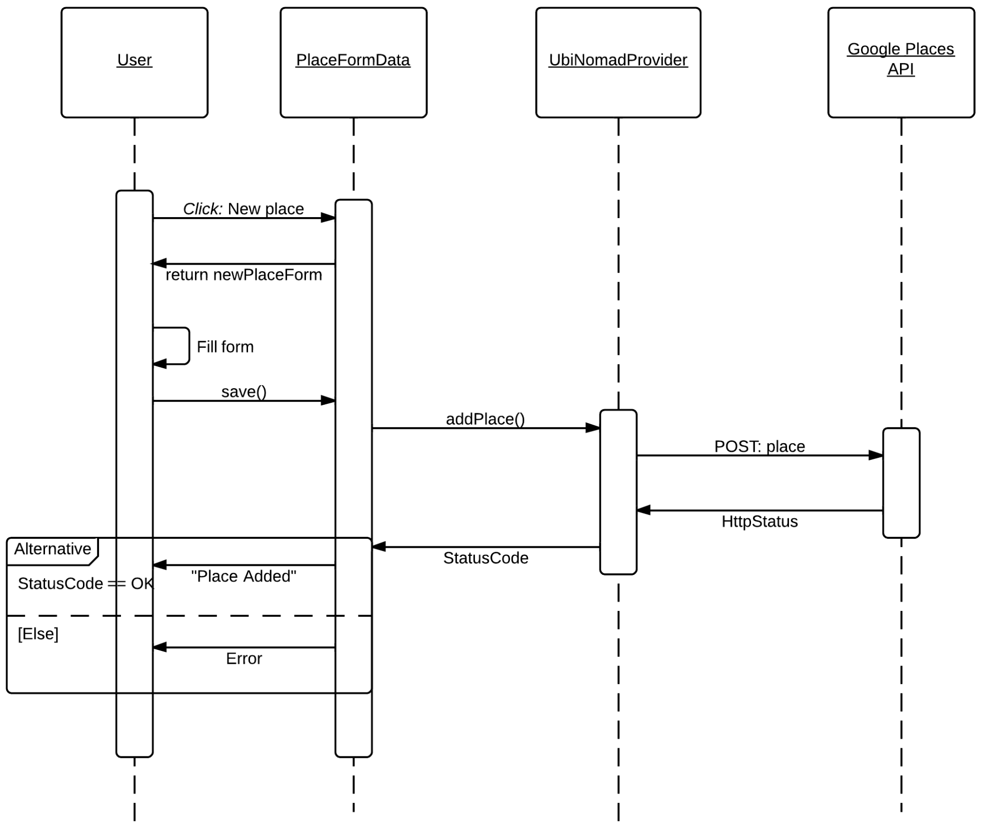


Figure 5.5: Sequence diagram - Create New Place

5.3.2 Delete Place

Considering we chose to save our own places as a Google place, we do not have much choice in how and when a place may be deleted. As long as the place is in a moderation state, the owner has the possibility to delete it. That a place is in moderation state means that a Google moderator will review the place and decide if it should be included as an official Google Place, displayed in Google Maps. However, while it is not an official Google place, it will still be available for applications using the same API key. If a place is accepted as an official place, it will no longer be able to delete it. Figure 5.6 describes what happens "on delete".

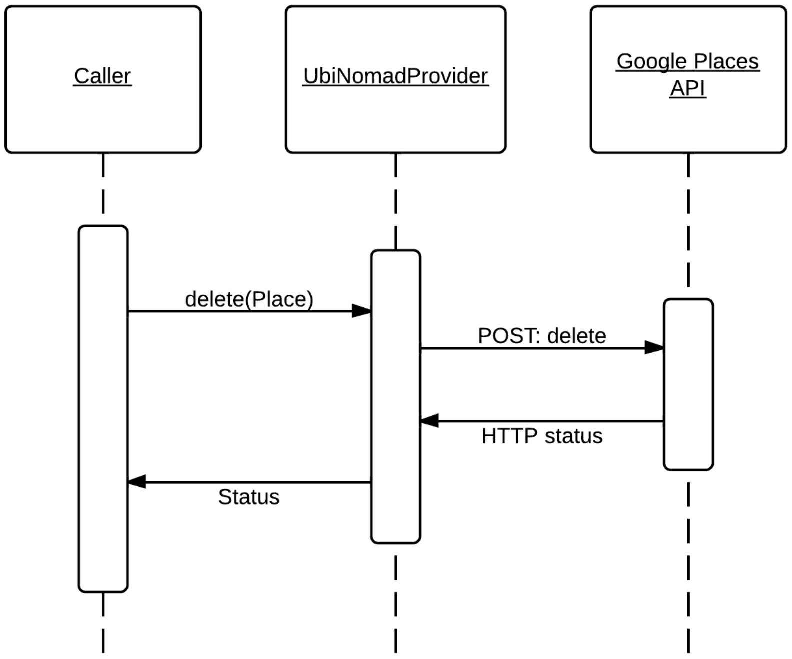


Figure 5.6: Sequence diagram - Delete Place

5.3.3 Edit Place

Edit a place is not supported as a stand-alone method, but may be archived by delete a place before adding it again.

5.3.4 Sync Places

In section 5.1.1 the architecture suggests we would use sync adapters, but by using that architecture as base we implemented an easier solution. A place is stored in the database after it is checked into the first time. By pressing a button a thread will start and traverse every stored raw place. For each raw place, it will find the corresponding place at the provider, compare them, and if they differ, update the current place, as seen in figure 5.7.

The reason behind using this method instead of sync adapters is that we did find implementing sync adapters harder than expected. In addition, we do have a way to get open sessions, which means connecting to the providers will not be hard.

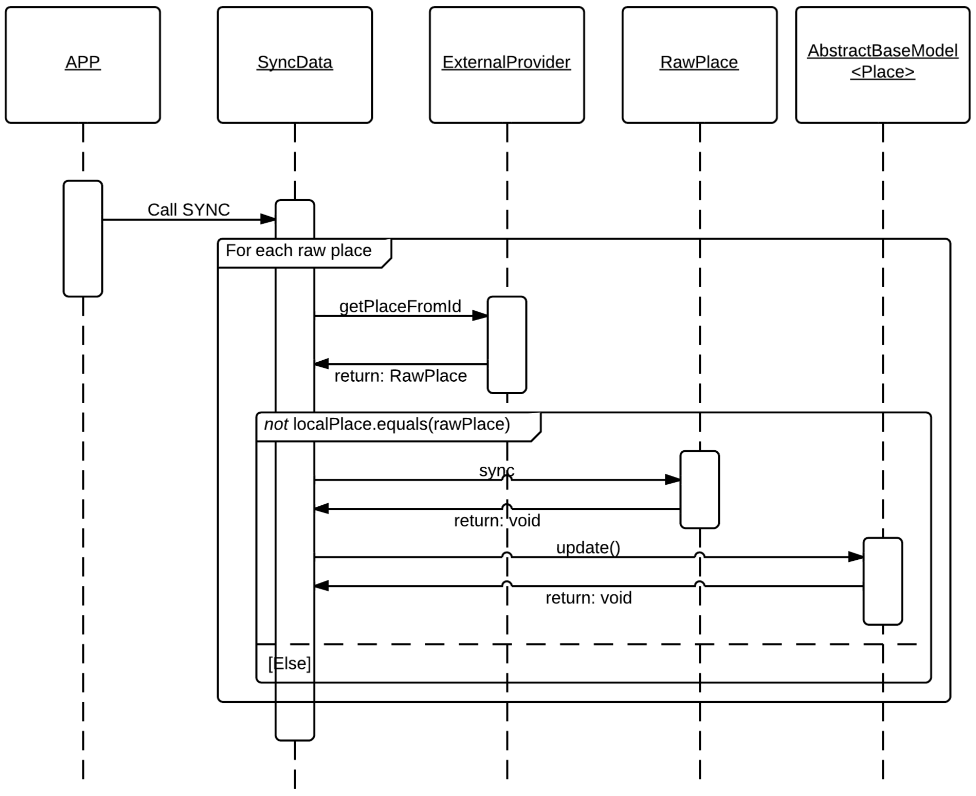


Figure 5.7: Sequence diagram - Sync places

5.3.5 Place From External Source

In UbiNomad we need to be able to get places from external sources. This may be achieved in multiple ways and with different purposes. In some cases we will be interested in getting a specific place from one specific source, while in other situations we would be interested in getting all places located nearby (section 5.3.6).

To be able to get a specific place, the application needs to have a reference to that place. Using the reference and the provider, we will be able to find the place by using the right external provider and call the findByReference method. The reference is a unique identifier, usually the id field. The request needs to be done in its own thread, because Android does not support network operations in main thread. (Figure 5.8)

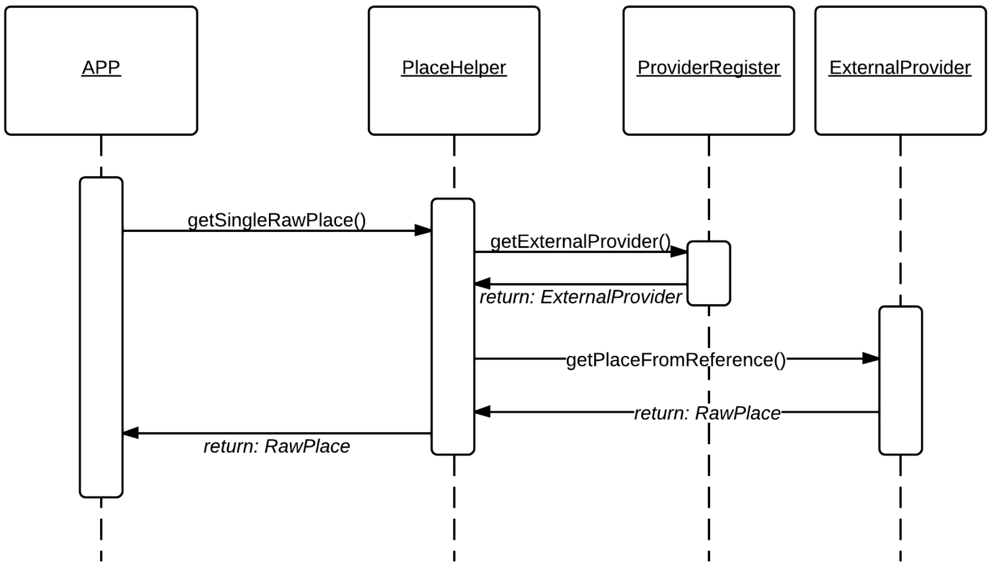


Figure 5.8: Sequence diagram - External place by reference

5.3.6 Nearby Places

An important feature UbiNomad is able to support, is listing nearby places. The way that is done in the library, as seen in figure 5.9, is by implementing getNearbyPlaces in each provider, and call every nearby method from another method. The application will call this method through an ExternalDataListener, where every external provider will answer by calling a method found in

ExternalDataListener. We figured that was the best place to put the function, because in that way it will be easy to expand the library and be able to change just a single method in a provider if it should be out-dated. Each call to an external provider will run in its own thread. By making every provider act on its own, we will not need to wait until all the providers have done their respective calls before returning places to the application.

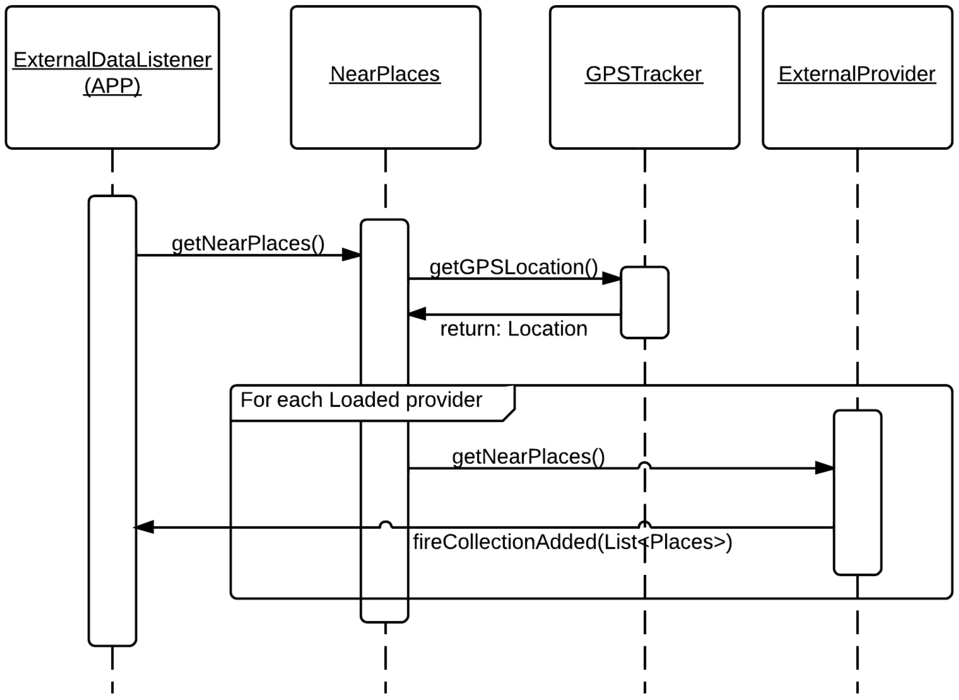


Figure 5.9: Sequence diagram - Nearby places

5.3.7 Update Current Place

Update current place, also called a check-in, is also an important feature that UbiNomad supports. The library supports the feature by using a method in a help class, as seen in figure 5.10. That method is able to find the active user, and set that users' check in place. The reason we want to do it all in in the library is because we want it to be easy to use. A developer should only need to call one method to check in a user.

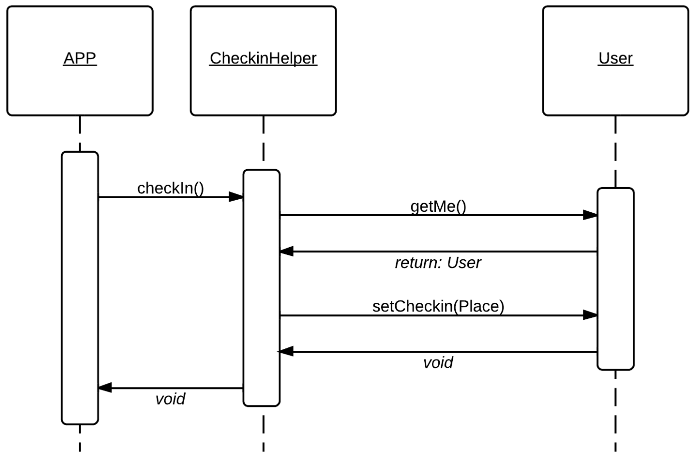


Figure 5.10: Sequence diagram - Place check in

5.3.8 Get Current Place

It will also be important that the applications are able to use the users' current check-in location. As seen in figure 5.11 this is done by using the CheckinHelper in much the same way as the previous use case.

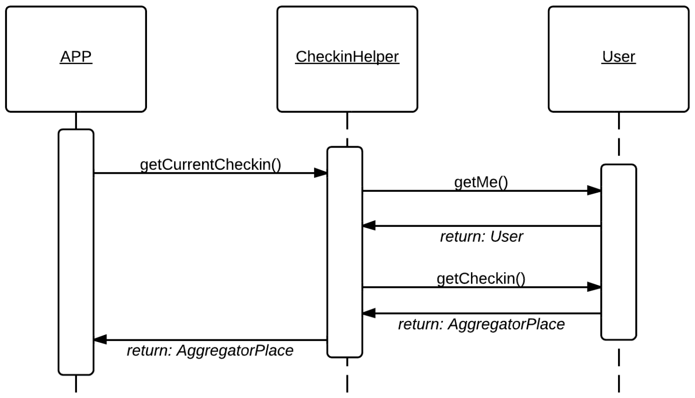


Figure 5.11: Sequence diagram - View current place

5.3.9 Map Places

To be able to share places across multiple social networks, UbiNomad needs to know which places are corresponding; for example which place on Facebook corresponds to a Google place.

To make that possible we created a place type called AggregatorPlace. The AggregatorPlace is used to map different places together. We will always be referring to a place as an Aggregator-Place in the library, because the AggregatorPlace consists of RawPlaces that correspond to that place, and are in fact the same place, just represented on different social networks. To be able to map those places together automatically would be a real challenge, and would probably be its own master thesis. Therefore, we chose to give that responsibility to the users. As earlier said, this section will not show how this is done in the manager, but focus on what will happen in the library. We will assume the application has already managed to pick a RawPlace and an AggregatorPlace, which are going to be mapped together. (Figure 5.12)

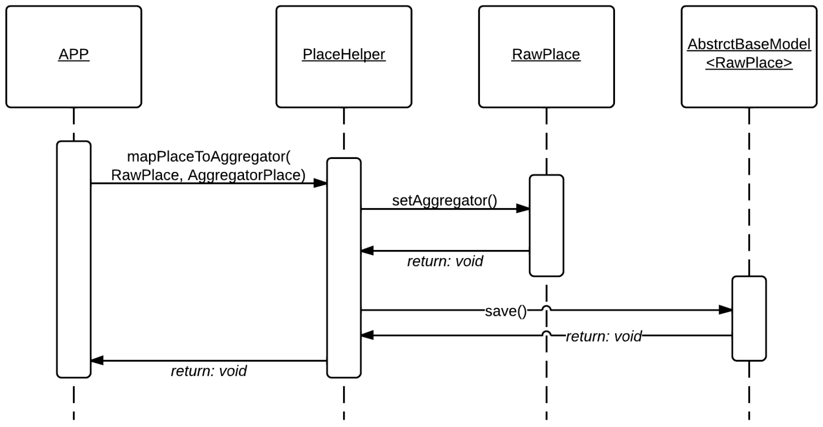


Figure 5.12: Sequence diagram - Map corresponding places

Chapter 6

Implementation

While we in chapter 5 presented an abstract view on how the different components of the UbiNomad Library were interacting with each other, will we in this chapter present a more detailed view of what is happening inside each component, and how the different providers are implemented. The code for the final solution can be found at https://github.com/vegaen/ubinomadlib.

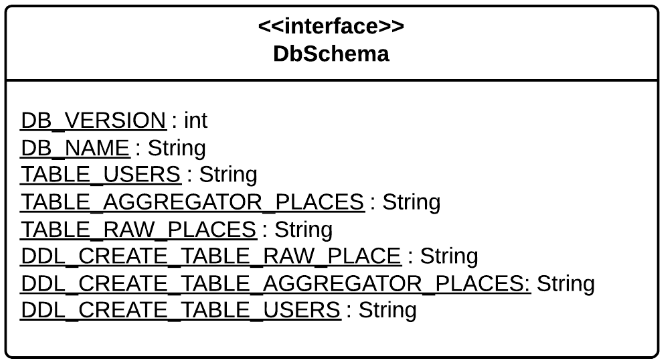
In section 6.1 we will describe how the database is defined in the library, and how it interacts with the content provider in section 6.2. Section 6.3 will describe how the classes are organized to make use of the database, before we, in section 6.4, will discuss a few components which are reusable by the applications depending on UbiNomad. Section 6.5 will in detail describe how the external providers are communicating with their respective sources. Section 6.6 will describe how we solved the problem of parsing JSON objects to Java objects.

6.1 Database

The UbiNomad Library is using a SQLite database locally on the device. It contains a table with users, a table with aggregator places and a table with raw places. A database in android is usually application specific, but the use of a content provider makes it possible to share it with multiple applications. How the content provider is used in this application is described in section 6.2.

When an application is starting, it will check if it is up-to-date based on database version. If it is outdated the application will upgrade it using the onUpgrade method in in "UbiNomadOpen-

Helper", which extends SQLiteOpenHelper. Through this class we are also able to override the onCreate method, which determines how the database is built if it is not present. To make the code easier to read and ensure a consistent database for the library, we created the interface "DbSchema". DbSchema contains the database version number, the name of the database file on the disk, and the construction queries for each database table (table 6.1).



<<interface>>
DbSchema
DB_VERSION : int
DB_NAME : String
TABLE_USERS : String
TABLE_AGGREGATOR_PLACES : String
TABLE_RAW_PLACES : String
DDL_CREATE_TABLE_RAW_PLACE : String
DDL_CREATE_TABLE_AGGREGATOR_PLACES: String
DDL_CREATE_TABLE_USERS : String

Figure 6.1: Database Schema

To be sure we are consistent with the database name throughout the implementation, we did also define "UbiNomadContract", which consists of interfaces for all the tables with column names and column order. By introducing those interfaces it was easier to avoid mistakes while constructing the content provider and data models. It also made it easy to modify the tables, because all changes were bound to one place.

6.2 UbiNomad Content Provider

A content provider is responsible for delivering data to an application. By encapsulating the data source the application does not longer need to know how the data is stored, whether it is a SQLite database or a it is gathered from the Internet, the application does not care. Every content provider needs to implement a few functions: insert(), update(), delete(), getType(), and query(), as seen in figure 6.2. By using this interface, applications are able to use data from the provider.

As discussed in appendix A.3, a content provider needs to be registered in an applications

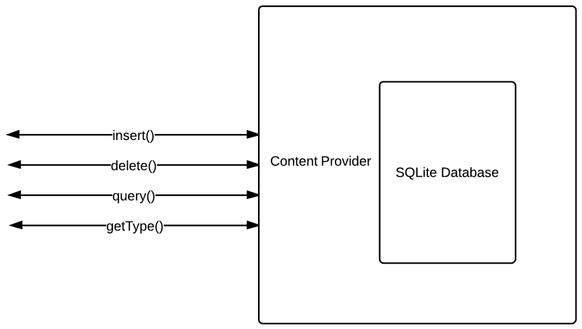


Figure 6.2: Content provider

manifest. Because a library project cannot be installed, we need to register the content provider another way. We chose to register this content provider to UbiNomad Manager. By doing this a user is required to install UbiNomad Manager before another application depending on UbiNomad could run.

6.3 Database Class Model

This section will describe how the objects are interacting with the database, how the implementation has developed, and how we have designed the models to be easy to extend.

We started off implementing the methods to get access to the database in every model, since it would have different return types and argument types depending on which class it was an instance of. After a short while we discovered those implementations where pretty similar so we decided to create an AbstractBaseModel, which implemented all the methods we used to communicate with the database. Using the type variable notation we were able to use the correct type for each method. By implementing this way, we only need to override the method that casts the object to and from an android.database.Cursor, to be able to do simple save and retrieve operations. The implementation makes it easy to change the methods used to communicate with the database, since it is used in multiple models. It will also be easy to extend the framework as one does not need to add much code. Figure 6.3 describes how the different classes are tied together.

When doing database operations, the methods will convert the object into a "ContentValues" object and send content values through the content provider. The content provider will then be responsible to get, insert or update data.

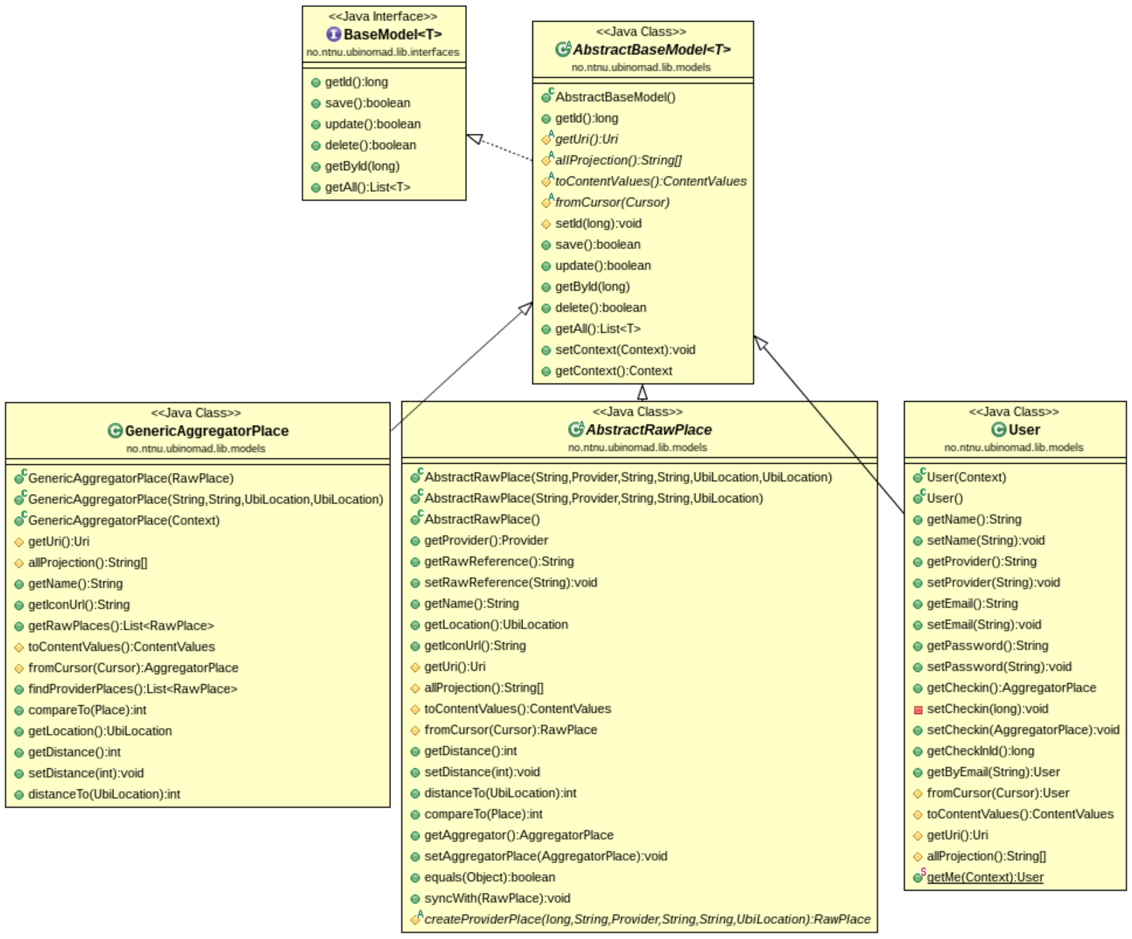


Figure 6.3: Database Class Models

6.4 Components

This section will describe components implemented in UbiNomad Library, that are reusable for applications depending on UbiNomad Library.

6.4.1 UbiNomadFragmentActivity

To have an easy setup for the developers, UbiNomadFragmentActivity is made available. The UbiNomadFragmentActivity includes a standard setup, including a main view, login view (described in section 6.4.3), and a menu system. It is not required to extend this activity in the main class, but it is recommended when getting started with the system.

6.4.2 PlacePickerList

The PlacePickerList is a list specialized to display places. Each element in the list is displaying a picture of the place, its name, an icon telling which provider we got the place from , and the distance to the place. The order of places is determined by how close the place is to your location.

The PlacePickerListAdapter is placed in the component package of the library project. This makes it possible for every application, depending on the library, to make use of the list adapter.

6.4.3 Login fragment

The login fragment handles login sessions to multiple providers, to make it easy for developers to include a login service with their application. The fragment includes a login layout that shows login buttons for the social networks included in ProviderRegister.

6.4.4 Location service

Unless you could tell where you are at a specific moment, getting location data would be useless. We therefore implemented a location service using the GPS in the device. The class "GPSTracker" makes use of Android's location manager and registers both a the network provider and GPS provider. When requesting a location it returns network location, or if the G PS is enabled it will return the GPS location.

6.5 External Place Providers

This section will present a few scenarios where we need data from an external data provider. For each scenario we will describe how each place provider gets data from its source. Most of these scenarios are listed in the use case section in architecture (section 5.3). All the providers have defined places different, and the definitions are listed in chapter 3.

6.5.1 Log in

To make it possible for the user to interact with a social network, they have to be able to log in. Each network have a different approach.

Facebook

To be able to log into Facebook, we included Facebook SDK for Android. The SDK supports a session manager, and is able to find an open session. If the session is closed, the SDK includes an AuthButton, which we included in the LoginFragment. When the AuthButton is clicked, the SDK connects with the native Facebook application to execute the login process. By using the native application the user is not required to write its credentials, only confirm that the application is allowed to use basic user information. However, if the native application is not present, a webview is shown, requesting user credentials, before it proceeds the login sequence.

Foursquare

Foursquare is using OAuth 2.0 as authorization method, and to be able to use it with Android, we had to include "foursquare-oauth-library". This library includes methods to be able to get an authentication token. It does not include any method to be able to figure out if a session already exists, hence we had to implement a singleton FoursquareTokenStore by our self, which is able to hold the authorization key if set. If an authorization key is not available the imported library uses the native Foursquare application to log in, returning an authorization key which is stored in FoursquareTokenStore. If the native Foursquare application is not present, the user is redirected to Foursquare application download page in Play Store.

Google

Instead of making a massive API, as Facebook has done, Google has divided its services into multiple APIs. To be able to use Google Places there is no need to login. The only thing required is that the developers register a Google application in "Google developers" to get an API key. Using this key an application is able to connect to Google Places. However, to be able to connect with Google+, the user is required to log in.

To be able to log into Google+, we needed to include google-play-services_lib, which is available through the Android SDK. By calling the method connect() in the library, the user is prompted for which account he will be using to complete the sign in.

6.5.2 List places near a position

This scenario is used when the user is about to check into a place, and we are listing the nearby places.

Facebook

As previously stated, Facebook uses a session manager. By using an open session it is possible to search for places near a position. The request returns a list of GraphPlaces, which is an object defined in the Facebook SDK. To be able to use these places in the library, we converted them to an instance of GenericRawPlace object. A list of the converted places is then returned to the caller.

Foursquare

To get Foursquare venues, we have to use a HTTP call. Calling `https://api.foursquare.com/v2/venues/search` will return a list of places, but we have to specify a few parameters. One required parameter is the "oauth_token" parameter, which identifies the user. In addition "v", which stands for version, has to be present. This parameter is only a date formated YYYYMMDD. To make the request know the location we are interested to get venues near, "ll" has to be defined. The "ll" parameter is formated "longitude,latitude".

The HTTP request is responded with a JSON object containing a list of venues. The JSON object is then paresed to a list of places (see section 6.6), before it is returned to the caller.

Google Places API

In the same way as Foursquare, Google Places API uses HTTP requests to get a list of places. The URL used for Google Places is `https://maps.googleapis.com/maps/api/place/nearbysearch/`

json. The required parameters for this search are "key" and "location". The key parameter identifies the application, while the location parameter represents the center of the search area. As default Google Places ranks places by relevance, which we found to be a hassle in the library, because Google Places returns a list of places with at most 20 places. That meant that if there are more than 20 places within the search area, it is a high probability the place the user wants to check in to is not in the list. By ranking by distance, we needed to specify the types parameter, but every place near the location was returned if the type matched. The "types" parameter was defined by place types as string, separated by "|". We wanted to see all places near the location, which meant we included every but one type variable in the types parameter. The only type we excluded was the "establishment" type, which returned peoples homes.

The HTTP request is responded with a JSON object containing a list of places. The JSON object is then parsed to a list of places (see section 6.6), before it is returned to the caller.

6.5.3 Get Single Place

The library is able to get a single place from every place provider, which is useful while syncing places.

Facebook

Using an open session, we are able to get a single place using the GraphPlace id.

Foursquare

Using the URL `https://api.foursquare.com/v2/venues/{venueId}`, we are able to return a venue as a JSON object by defining the parameters, oauth_token and v, as described in the previous section. The JSON venue will be parsed into a Java place to make it useful by our API.

Google Places API

Using the URL `https://maps.googleapis.com/maps/api/place/details/json`, we are able to return a place as a JSON object by defining the parameters "key" and "reference". Google

Places have not the possibility to get a place by its id, but the reference is unique, which makes it work in the same way.

6.6 Parse JSON Objects Into Java objects

Both Foursquare and Google Places returns a place as a JSON object. To easily be able to parse JSON objects to Java objects, there is a method in the Google API doing that, called parseAs. The method takes a class as parameter and by using the @Key annotation a variable is matched with its corresponding JSON key. When the JSON object in listing 6.1 is parsed as a Person object in listing 6.2, it will give us a Java Person object named "John Doe" at the age of 32.

```
{
  "name": John Doe,
  "age": 32
}
```

Listing 6.1: JSON representation of a person

```
class Person implements Serializable{

  @Key
  public String name;

  @Key
  public int age;
}
```

Listing 6.2: Java representation of a person

Chapter 7

Applications

In this chapter we will explain the different applications we developed in addition to the library. We developed three applications, the UbiNomad Manager(section 7.1) and two sample applications. One of the sample applications, the UbiChat application(section 7.3), was not fully developed because of time restrictions, but is included here to explain how the application was intended to use our library. The second sample application, UbiPost(section 7.2), was fully developed and takes full advantage of our API. These sample applications were developed for two reasons: to demonstrate how the API works and for us to evaluate the usefulness of our solution (chapter 8).

Each application has been described more deeply in the upcoming sections. In each application we will explain how they use the API, how the development process was and describe the application with screen shots. To see an example of how to create an application and how to use our solution, see the "Hello UbiNomad" example in appendix B.3.

7.1 The UbiNomad Manager

Together with the library, we developed UbiNomad Manager, which is supporting every functionality we implement in the library. The UbiNomad Manager is vital to have installed on the device to make use of the UbiNomad API. Note that at current time, to be able to use the application, location service has to be activated on the device. If it is not, the application will get a runtime exception at startup.

UbiNomad Manager is responsible for registering the content provider in its manifest. Because the provider has to be registered and the authority is unique, it was hard to find a solution where you are not bound to be dependent on one application.

The Manager does also support features making it possible for the user to explore every aspect of the API. If the UbiNomad Library supports a feature, it is possible to use it through the manager. Features supported by the manager are described in section 4.3 and section 5.3.

7.1.1 The Development Process

The development of the manager has been done in parallel with the development of the library. We started the development by creating a prototype of the application, which through multiple iterations was modified and improved, leading towards the prototype in figure 7.1. During the implementation step, the manager was one of our easiest ways to check whether or not implementations of new features in the Library were included and worked properly. We started the implementation by creating a basic GUI setup as close to the prototype as possible, while we implemented the first features in the library. Once a feature was implemented in the library, we found a way to present and make it possible to accomplish the feature through the manager.

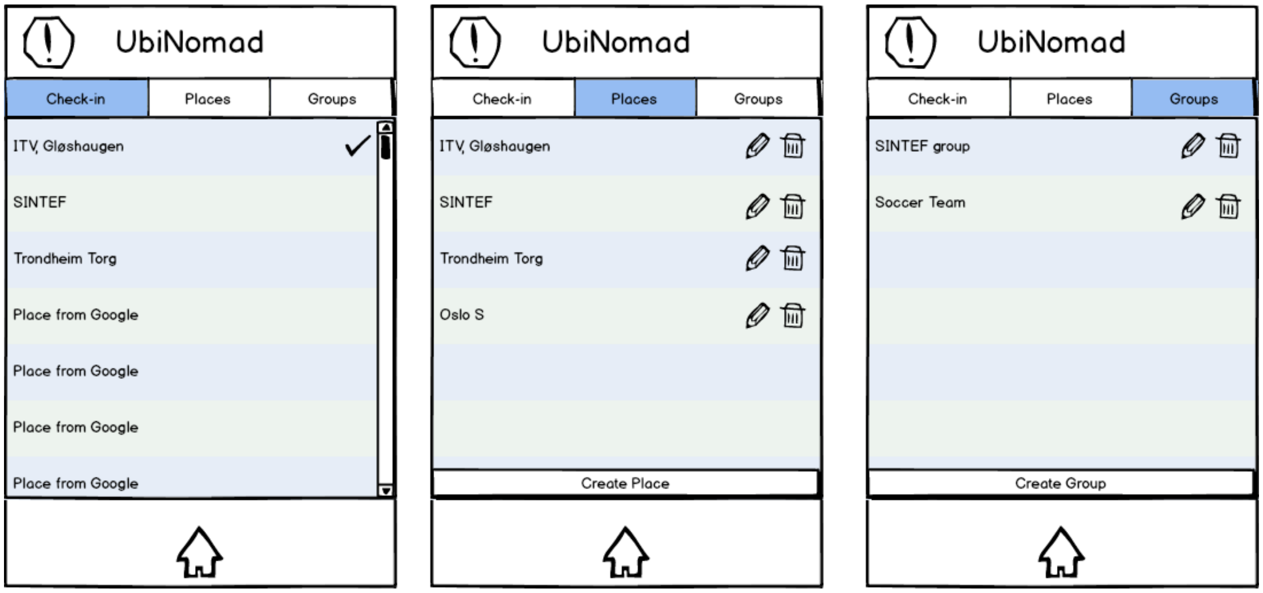
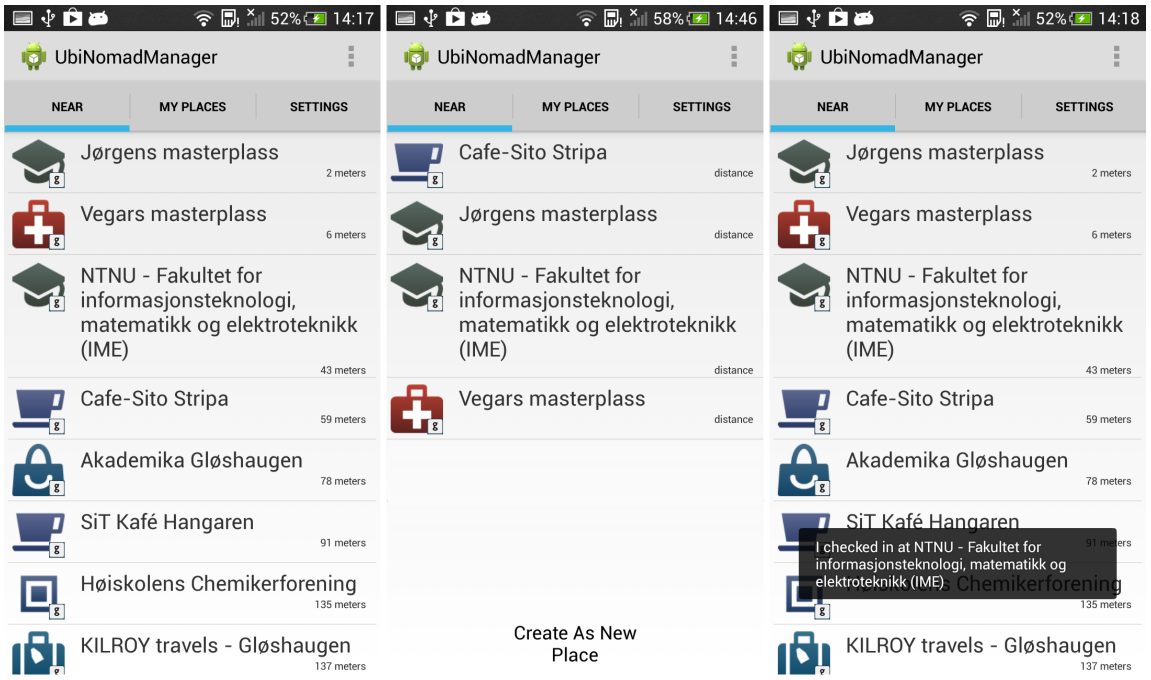


Figure 7.1: UbiNomad Manager Prototype

Throughout the implementation our goals were changed, through the agile development process. The changes made us alter the GUI to our final stage. Figure 7.2 shows the check-in process step by step. In figure 7.2a the UbiNomad Manager displays the list of the nearby places and the user press the place he or she wants to use as current place. Then in the next step the user assigns the place to an aggregator place (figure 7.2b) and gets an check-in confirmation, showed in figure 7.2c.


52% 14:17
UbiNomadManager
NEAR | MY PLACES | SETTINGS
Jørgens masterplass 2 meters
Vegars masterplass 6 meters
NTNU - Fakultet for informasjonsteknologi, matematikk og elektroteknikk (IME) 43 meters
Cafe-Sito Stripa 59 meters
Akademika Gløshaugen 78 meters
SiT Kafé Hangaren 91 meters
Høiskolens Chemikerforening 135 meters
KILROY travels - Gløshaugen 137 meters

(a) Near places

58% 14:46
UbiNomadManager
NEAR | MY PLACES | SETTINGS
Cafe-Sito Stripa distance
Jørgens masterplass distance
NTNU - Fakultet for informasjonsteknologi, matematikk og elektroteknikk (IME) distance
Vegars masterplass distance
Create As New Place

(b) Aggregator Places

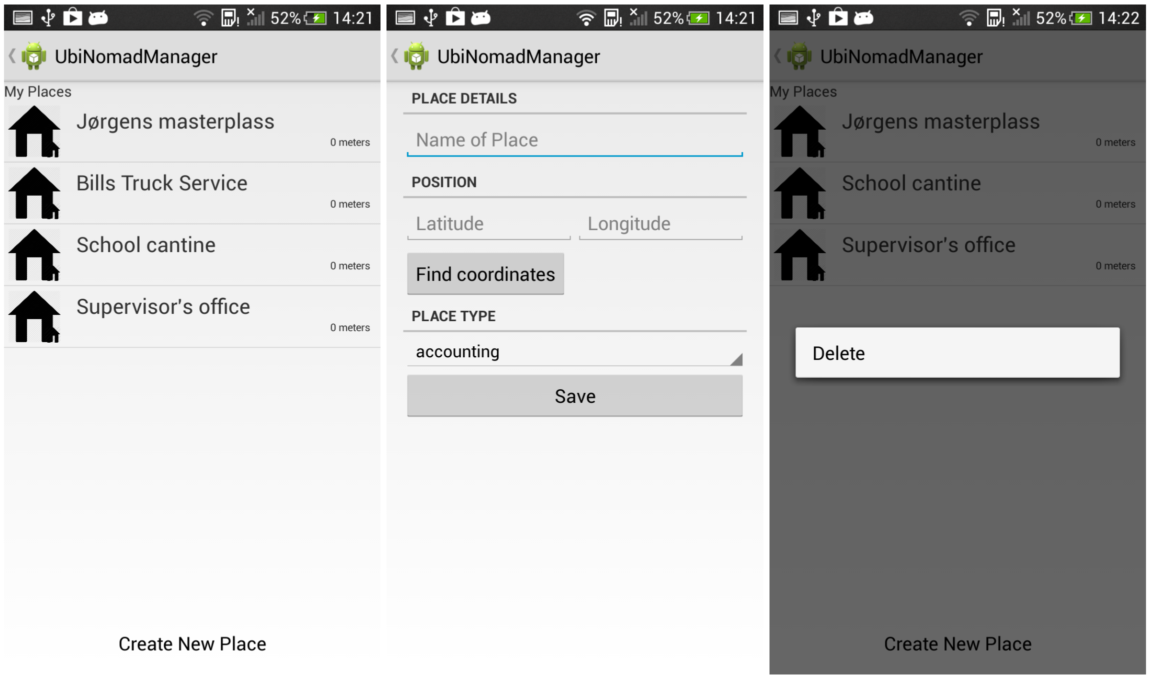
52% 14:18
UbiNomadManager
NEAR | MY PLACES | SETTINGS
Jørgens masterplass 2 meters
Vegars masterplass 6 meters
NTNU - Fakultet for informasjonsteknologi, matematikk og elektroteknikk (IME) 43 meters
Cafe-Sito Stripa 59 meters
Akademika Gløshaugen 78 meters
SiT Kafé Hangaren 91 meters
I checked in at NTNU - Fakultet for informasjonsteknologi, matematikk og elektroteknikk (IME)
Høiskolens Chemikerforening 135 meters
KILROY travels - Gløshaugen 137 meters

(c) Checked-in

Figure 7.2: UbiNomad Manager - Check-in process

Figure 7.3 show some of the actions regarding the created places. In figure 7.3a UbiNomad Manager displays the created places the current user has created. By pressing the "Create new place" button the system changes view to a "create place" form(figure 7.3b), where the user can type in the relevant information. Pressing the "Find coordinates" button, the system finds the user's current GPS coordinates and place it in the longitude and latitude fields. If the user wants to delete a place he or she must long press the place in figure 7.3a and a confirmation window is displayed(figure 7.3c).

In figure 7.4 we see the different settings in the UbiNomad Manager. Pressing the "Settings" tab takes you to the login screen, displayed in figure 7.4a. If the user presses the three dots in the upper right corner, the user gets the ability to sync and update the external places that is stored,



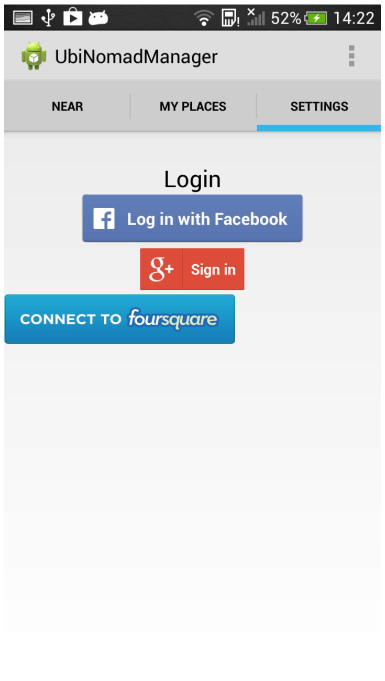
(a) View Users Created Places (b) Create Place Form (c) Delete Place

Figure 7.3: UbiNomad Manager - Created Places

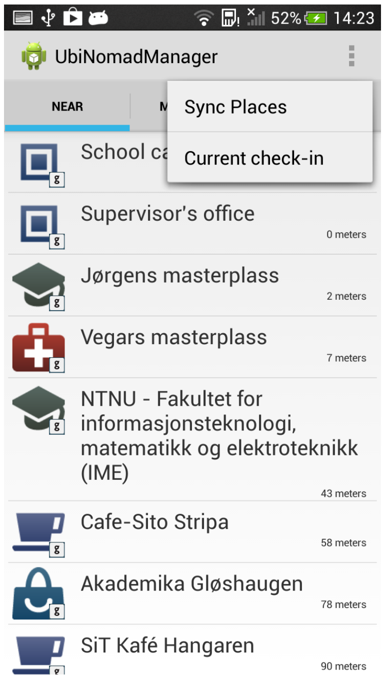
and also to see the current check-in (figure 7.4b).

7.2 UbiPost

UbiPost is an application that lets the user post his or her current place to different social networks at the same time. The supported social networks are Facebook, Google+ and Foursquare, and the place is posted as a "status update". The post process is done with just a few steps. First the user makes sure he or she has checked in at an aggregator place in the Ubinomad Manager, because UbiPost needs the user's current place to be able to post a status. Secondly, the user adds some additional text to the status post and press the "Post" button. Then UbiPost automatically extracts the raw places from the user's current place and uses them for each individual social network post. For example, the applications takes the raw Facebook place from the aggregator place and posts it to Facebook. Then the UbiPost application does the same procedure for the other social networks and raw places.



(a) Login Screen



(b) Sync and View Check-in

Figure 7.4: UbiNomad Manager - Settings

7.2.1 The Development Process

The idea for the UbiPost application came to mind during the "Design and Implementation" phase, while working on the UbiChat application, and the development of UbiPost took us about a week. While developing this application we moved much of the functionality from the application into the API, so other developers can access post functionality in their applications through UbiNomad.

7.2.2 Technology

The way this application is using the UbiNomad API, is by using the ProvideRegister to find the providers registered to the application, and call each of them and run the postStatus() method. It is also using the UbiNomadFragementActivity as main activity and the login fragment.

The postStatus() method in the API has the parameters "Message" and "AggregatorPlace". Message is a string that contains the status update, and by using the AggregatorPlace the method finds the appropriate RawPlace and adds it to the post. In Facebook this will be shown as "Status text - at RawPlace".

7.3 UbiChat

UbiChat was supposed to be a simple chat client where the user talked to nearby people, but we never got to finish the development of this application. The idea was that the users communicated with each other in a "chat room", and which chat room the user was a part of was decided by the user's check-in. The users could only talk to people that were checked-in at the same place. With this functionality we were taking advantage of the places in our solution and still not creating a too complex application, but unfortunately the application only became a prototype(figure 7.5). Despite that UbiChat was not fully developed, we got a lot of experience with Android development with this application.

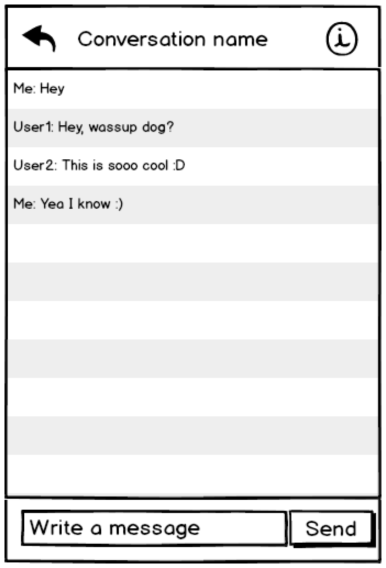


Figure 7.5: UbiChat Prototype

7.3.1 The Development Process

UbiChat was the first sample application we developed and it was developed early in our project. When we started on the development we had almost no experience with Android and we needed to develop an application many had done before us, and was well documented on the Internet. In the beginning UbiChat was intended to only be a normal chat application like all the others, with the user's location added to the messages, but during the development we wanted to make it a little different. We then came up with the idea that the users could only chat with people checked-in at the same place. Unfortunately the development of the application was stopped, and the application became outdated while we further developed on our solution.

7.3.2 Technology

UbiChat used a client-server model, consisting of a server and one or more clients. The clients connected to the server, and the server stored every connection in a pool of connections. When a client sent a message, the server would broadcast it to every client in the pool. The application made use of the API by using the login mechanism.

The way it could benefit by the UbiNomad API is that it could take advantage of AggregatorPlace mappings. If a user sends a message from a Facebook place, and the second user only uses Foursquare, he could still be able to figure out the place the message was sent from, based on the common aggregator place.

Chapter 8

Evaluation

In this chapter we will explain the evaluation process and the results. The evaluations were performed after the development phase ended. In this evaluation phase we looked into some evaluation criteria to see if our solution supported those and whether we achieved these criteria or not. With these criteria we have evaluated both advantages and disadvantages for our solution which is further discussed in chapter 9.

8.1 Evaluation Criteria

During this project we were free to decide which parts of the project task we wanted to focus on, and which requirements our solution should contain. Without any clear goals for the solution, this led to some changes during the development phase, as new ideas came to mind. Before these ideas were included in the requirement, we had to discuss it with our supervisor. He made sure we always were heading in the right direction, and redirected us if he did not agree with our suggestions.

To evaluate our project we have focused on achieving the requirements from chapter 4 and measured the usefulness of our solution. These tests have only been ran by ourself.

Our evaluation has three criteria:

1. Our solution shall achieve all the functional requirements

2. Our solution shall be extendable and modifiable

3. Our solution shall easily be implemented in an application

8.2 Results

The first criteria is that the functional requirements shall be supported by our solution. As seen in figure 8.1 all the requirements were achieved, except for the edit place requirement(section 5.3.3). The alternative action the user has to do instead is to delete the place and create a new.

Requirement	**Success**	**Alternative Action**
UbiNomad shall allow users to create places	✓	
UbiNomad shall allow users to edit places	✗	Delete and create a new place
UbiNomad shall allow users to delete places	✓	
UbiNomad should be able to get places from external sources	✓	
UbiNomad shall allow users to view their places	✓	
UbiNomad shall allow users to update current place from external sources and created places	✓	
UbiNomad shall allow users to map places together	✓	
UbiNomad should share created places between the users	✓	
UbiNomad should be up to date with places from external sources	✓	
UbiNomad should be able to display the right information about a place to the user	✓	
UbiNomad should be able to share current location with other applications	✓	

Table 8.1: Evaluation of the Functional Requirements

The second criteria was evaluated by extending the library with Foursquare. During the "Design and Development" phase we only implemented Facebook and Google Places as place providers, and by adding Foursquare after the development phase was over, we could evaluate the process of extending the library with a third place provider. We sat the time frame we meant an experienced developer with some knowledge should be able to complete the task on to eight hours, which is about one day of work. We thought that was a reasonable time frame because adding a new place provider is nothing that is done regularly, which means that there is no reason it should be done in a shorter period of time. However, we will encourage to add multiple place providers, which means it is not acceptable a developer has to spend multiple days extending the framework with an additional place provider.

In the validation phase of this project we implemented Foursquare, to validate if we had fulfilled the criteria above. We used in total eight hours to implement Foursquare, which was within the set time frame. What and how the implementation was done, is described in appendix C.3

To be able to evaluate the third criteria, we need to consider a few different aspects. The first thing we are able to measure, is how many lines of code we need to include in the application to be able to make use of the library. In our application using UbiNomad Library, we needed to include one line of code for each place provider, which is in our opinion within the acceptable boundaries. Secondly, we may measure if the library saves the developers for any amount of coding. As the library supports an API supporting different features, which are implemented for every place provider in the library, the developers will have a lesser amount of code they need to write and maintain, as long as they are going to use multiple social networks. UbiPost, the sample application we made, had in total three lines of functionality needed to be implemented, everything else was UI setup.

The third aspect of this criteria is we have focused on is, how easy and time consuming it is to set up the development environment. Facebook SDK, Android SDK, Foursquare, and UbiNomad Library are all Android library projects, which means that we are not able to create a binary file (jar) including resources[3], and they all need to be imported as projects (in Eclipse) or modules (in Android Studio). UbiNomad Library will always include these libraries when downloading the Git-project. By including the libraries, it will also make it harder to keep these external libraries updated.

The fourth and last aspect we needed to evaluate was how complete the UbiNomad Library is. Does it support the functionality the developers need? By this time the answer to that question would be no. The UbiNomad Library supports only a limited amount of functionality provided by the providers. However, it is still possible to write provider specific code within the application, because the application depends on the UbiNomad Library that depends on the provider specific libraries.

Even though there are a few drawbacks with the last criteria, most of those are things beyond what we are able to control, we will say we are close to fulfill the last criteria as well.

Table 8.2 compares how different aspects are done with and without UbiNomad, and supports the "Easy to implement" requirement.

Aspects	**Pre**	**Post**
How to initialize a social network	Include a library for this specific network, add the variables and functions according to that networks documentation	Add one line to initialize the application
Code complexity based on the number of social networks implemented	Seeing that each network has its own method of doing the same thing as another, which makes the code quite complex when trying to handle both in the same application.	Because the developer is working against one library, the code will not be more complex with multiple social networks. It is also possible to add an additional network during the project, without having to modify the code
Features supported	Supports all the features included in the social network library	Supports is limited to features implemented in UbiNomad. However, there is possible to extend with more features.

Table 8.2: Compare criteria pre and post UbiNomad

Chapter 9

Conclusion

This chapter will wrap up the project. In section 9.1 we will discuss and reflect upon the result of the project. In section 9.2 we will describe features we did not have time to implement, while we in section 9.3 will present features we think will improve the UbiNomad Library.

9.1 Discussion and Reflection

The goal with this master thesis, was to make it easier for developers to create location-based applications across social networks. We have made a tool, called UbiNomad, trying to solve that problem. To figure out if we have fulfilled that goal there is one question that needs to be answered: Would a developer benefit by using UbiNomad?

This project has shown us that UbiNomad, as of today, is not good enough to be a complete middleware between location-based Android applications and social networks.

As seen in the evaluation part, we are able to fulfill most of the criteria. The great parts from the evaluation is that UbiNomad is easy to implement in an application, and as long as more than one social network is implemented in an application, it will lessen the amount of code.

To set up the development environment is not as easy as it should be, but this is something we are not able to control at the moment. However, if you were to implement multiple social networks in a normal application the setup would be at least equally complex.

We have had some huge advantages using the UbiNomad Library, and there should absolutely be a market for it. However, there are a few drawbacks. The first drawback is that the limited functionality supported by UbiNomad does not encourage use, as of today. However, with just a little bit of work the functionality may be hugely extended, which would make UbiNomad a more likely alternative. In addition, all of the SDKs we have used during the development are in constant change, which means that on the day of release, UbiNomad may already be outdated.

This project has only been a master project and more work needs to be done before the final solution will help the developers as first intended. We hope at least that our solution can be used as a base for further development, and that it some day will be useful for other developers. We have gained a lot of experience through this project, and we hope that we were able to share them with other developers through this master thesis. There still exists some work for UbiNomad, and we have written section 9.3 to contain some of the ideas we have for future work, and to make it easier for future groups to continue where we ended.

9.2 Limitations

This section will describe a few features we wanted to implement, but did not have enough time to do. This includes:

- Test With Other Developers
- A widget making it easier to check in to places
- Usability improvements in the UbiNomad manager
- The possibility of creating groups
- Integrate UbiShare
- Be able to share aggregator places

9.2.1 Test With Other Developers

We have not been able to test the API on other developers than ourself, because the development process took too much time and setting up user tests for an API was not that simple as an single application would have been. It would require too much time for the users to test our solution.

9.2.2 Widget

The previous UbiNomad implementation contained a widget. We did not manage to create one ourself for our solution, but it will be of big help for the user if a widget can be developed for our solution. With this widget it will be easier and faster for the users to check-in at places and the widget can also have the functionality to map raw places together by create aggregator places.

9.2.3 Usability Improvement of the UbiNomad Manager

The manager applications has gone through different changes trough the development phase, but it has never been exposed to a usability test. We focused only on achieving the requirements and not the usability. This application needs some improvements both regarding graphics and usability.

9.2.4 Share Aggregator Places

At this point, our solution only supports sharing of created places and not the aggregator places. This is a task that needs high priority because it will increase the usability of UbiNomad. Now the user has to create his or her own aggregator places and can not use the other users' mappings.

9.2.5 Groups and UbiShare

To be able to make it possible to create cooperative applications to this API, the possibility to create groups is necessary. Groups will make it possible to share places with a specific group of people, which will enhance the privacy.

To be able to also share files within a group, there would be an idea to implement UbiShare with UbiNomad.

9.3 Future Work

During the time we have worked with the project, we have gained more knowledge of the scope UbiNomad operates in. Based on more knowledge we got ideas on how UbiNomad could be extended and improved in the future. These features and ideas include:

- Keep UbiNomad updated with Facebook, Foursquare and Google Places. Always new features and constant changes in the API.
- Extend with other social networks
- Automatic update of current location
- Location of friends
- Place search
- Distribute library as binary file
- Automatically map places together

9.3.1 Update UbiNomad

Facebook, Foursquare and Google Places are in constant change and so the next generation of UbiNomad has to be up to date. The changes can cause UbiNomad to crash and not function properly. The place definitions also need to be up to date in case of changes.

9.3.2 Add Social Networks

If new social networks with location sharing features come to market, UbiNomad would benefit if they were added to the solution. We want the library to cover as many place providers as possible.

9.3.3 Automatic Check-in

Beside the widget, there can also be created a function that automatically change the user's location as he or she is on the move. Making check-in automatically is not an easy task, because you have to be able to make guesses about whether a person is actually staying at a place, or just passing by. You will also have to guess which of the nearby places the person is staying at. Even though this task is hard, it would be a nice to support this feature in the library.

9.3.4 Get Friends' Location

An important feature that may open a lot of possibilities for new applications depending on UbiNomad Library, is the ability of getting a friends' location. This includes getting a single friend's location and all his friends' locations.

9.3.5 Search Places

It would also be of good help if the user was able to search for a specific place in the list of nearby places. This list can be quite big in central regions, and the place the user is looking for is not always at the top of the list. This feature has a much lower priority compared to some of the other features.

9.3.6 Distribute Library as Binary File

Right now it can be a hassle to set up the development environment, because we have to import the libraries as projects, instead of binary files (.jar). This is not possible at the moment as stated: "A library cannot be distributed as a binary file (such as a JAR file). This will be added in a future version of the SDK Tools."[3] However, it will be possible in the future, and should be done once it is possible.

9.3.7 Automatically Map Places Together

This project depends on the users' ability to be able to map corresponding places together. This process may be troublesome because every user has to this, and may be a barrier to get users to use the API. This may be solved by using an automatic place mapper, that automatically finds the same place across social networks.

Appendix A

Android Platform

As this project is all about mobility, it was obvious that we needed to write for a handheld device. Openness is without a doubt an other important aspect of this project, because the UbiCollab project is open-source. All UbiCollab sub-projects are written for Android, and because Android development was a requirement set by our supervisor, the choice of platform was set to Android.

Android is an operating system based on the Linux kernel, primarily built for mobile handheld devices, such as smartphones and tablets. The platform is open source and released under the Apache License by Google [16].

This appendix will briefly describe how the android platform works, including a description of the components we will use in the UbiNomad library.

A.1 Android Architecture

Figure A.1 shows the Android software stack, which may be represented by five sections divided into four main layers[2].

A.1.1 Linux Kernel

Linux 2.6 appears on the bottom of the Android software stack. This layer provides the system with basic functionality such as memory management, process management and device management. The kernel also contains a massive amount of device drivers and handles networking.

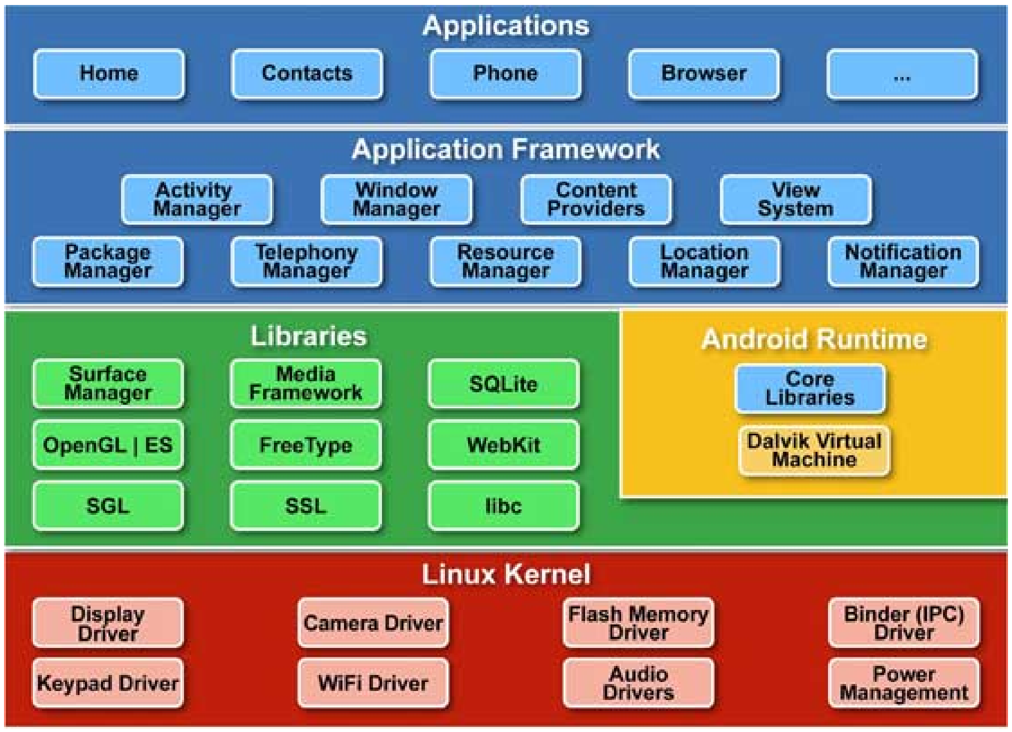


Figure A.1: Android Architecture [2]

A.1.2 Libraries

The libraries section contains a vast array of open-source libraries making it possible to do standard operations. These operation includes storage and sharing of application data with a SQLite database, creating a secure Internet connection through SSL, and play and record audio and video.

A.1.3 Android Runtime

This section provides a key component to the android stack called Dalvik Virtual Machine which is a specially designed Java Machine, optimized for Android. Every Android application runs its own instance of the Dalvik VM. Dalvik VM is described as a "just in time" compiler, which means it does the compilation at run-time. In Android 4.4 Google is playing with the idea to use ART runtime instead of Dalvik. ART runtime is an "ahead-of-time" compiler, which means it does the compilation prior to execution. ART is not enabled by default, but is possible to change to in Android 4.4.

In addition to provide a virtual machine, the Android Runtime section provides a set of core libraries which makes it possible to use standard Java programming language to write Android applications.

A.1.4 Application Framework

The Application Framework layer provides a set of various high-level services, such as Content Providers and Telephony manager. Application developers are allowed to make use of these services through a Java interface.

A.1.5 Applications

The top layer in the Android software stack is the application layer. Application developers will only be able to write and install applications on this level.

A.2 Android Activities

An activity is a single, specific thing a user can do. One example may be listing places, while an other may be adding a place. Almost all activities are using user interaction, so the Activity class makes it possible for the developer to set a user interface through the setContentView(View) method [1].

A.2.1 Fragments

Starting with Android 3.0, Activities implementations could better modularize the code by making use of Fragments. The fragments made it possible to build user interfaces that would scale better for small and large screens.

A.2.2 Activity lifecycle

Activities in android are managed as an activity stack. When a new activity is started, it becomes the running activity and is placed on top of the stack. The previous activity will remain at second position in the stack, and will be the running activity when the new activity exits.[1]

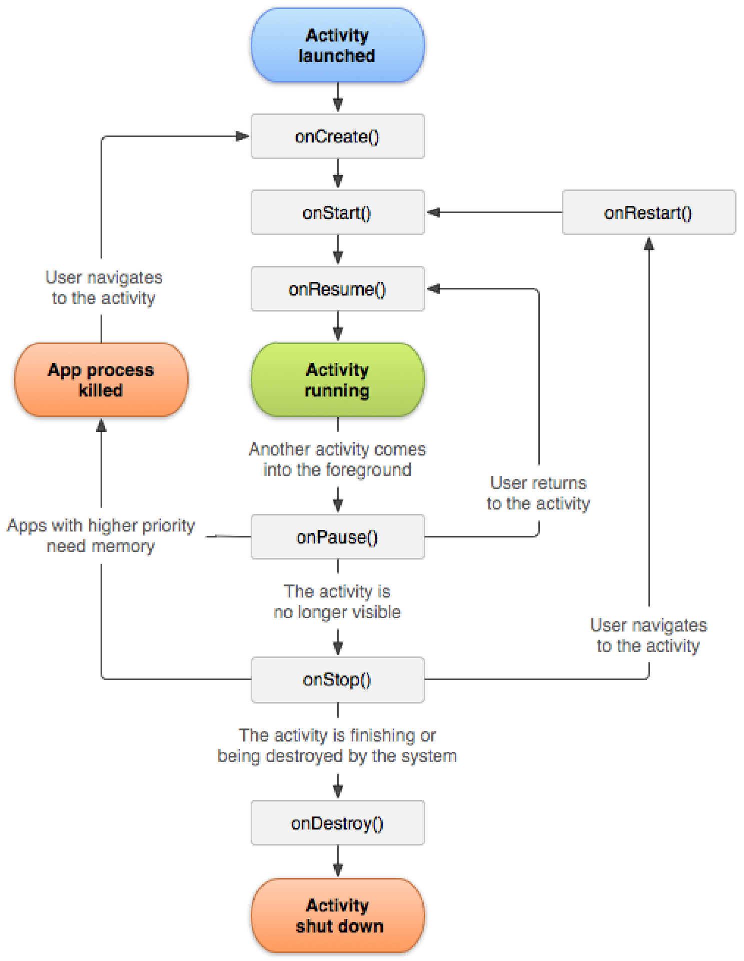

Activity launched
onCreate()
onStart()
onRestart()
onResume()
User navigates to the activity
App process killed
Activity running
Another activity comes into the foreground
User returns to the activity
Apps with higher priority need memory
onPause()
The activity is no longer visible
User navigates to the activity
onStop()
The activity is finishing or being destroyed by the system
onDestroy()
Activity shut down

Figure A.2: Android Activity Lifecycle[1]

Activities have, like other Java programs, a starting main method, but in an Activity this method is called onCreate instead of main. Figure A.2 shows the lifecycle of a standard Android activity.

An activity has essentially three states:

Active/Running An activity in the foreground of the screen

Paused If an activity has lost focus but is still visible, it is paused. A paused activity is completely alive, but can be killed by the system if it gets extremely low on memory.

Stopped If an activity is completely obscured by another activity, it is stopped. It still retains all state and member information, although it is no longer visible to the user because its window is hidden. It will often be killed by the system when memory is needed elsewhere.

A.3 Content Provider

Content providers are provided by the *Application Framework* and are used to manage access to a structured set of data. A content provider encapsulates data which will also provide data security. Content providers are the standard interface used to connect data in one process with code running in another process.[12]

Within a content provider the data can be stored in numerous ways. For instance the data could be stored in a SQLite database, on a server on the Internet, or as files. However, because a content provider encapsulate data, the process using the data is indifferent about how it is stored.

Whenever a process wants to access the data in a content provider, it will need to use the ContentResolver object found in the applications Context. The Content Resolver provides a way to communicate with the provider as a client. The provider receives data request from the client and returns the results through the content resolver.[12]

There is no need to implement a content provider if the application is not intended to share data with other applications. However if you want to perform custom search suggestion in the application, or want to share complex data or files with other applications, a content provider implementation is required.[12] To be able to use a content provider, it needs to be registered through the manifest in one application using an unique identifier, called authority.

Content providers that manage audio, video, images, and personal contact information are natively included in Android and are accessible to any Android application.

A.4 Sync Adapter

Sync adapters in Android provide the interface that transfer data between a device and a server. According to how the triggers and scheduling are set, the sync adapter will run the code in the sync adapter component[13]. Sync adapters may be used in different ways, data may for example be transfered form a device to a server as backup, or transfered data from a server to a device will make it useful even if the device is offline.

Sync adapters run asynchronously and should be used to transfer data regularly. It is not intended for real-time data transfer.

A.5 Widgets

Opposite to an application, a widget is an element that may be included in the home screen customization. It may be a lightweight version of an application, presenting the most important data and functionality, and is accessible right form the home screen. Widgets may be resized to tailor the amount of information viewed to an amount the users' preference.[4]

Appendix B

Getting Started with UbiNomad Library

This chapter is created to make it easier for other developers to continue developing on our solution. If any changes occur after this thesis has been submitted, go to `https://github.com/vegaen/ubinomadlib` to find the updated version, which also includes some pictures.

B.1 Install Google Play Services

1. Open Android SDK Manager
2. Install *Extras* -> *Google Play Services*

B.2 Prepare IDE

This section will present a short tutorial on how to make Eclipse ready for UbiNomad Library

1. Download the code from Github: `http://github.com/vegaen/ubinomadlib`
2. Import an existing Android application
3. Select root directory
    - Google Play Services: *[android-sdk]*/extras/google/google_play_services/libproject/google-play-services_lib

- Other dependencies: *[UbiNomadLib]*

4. Select projects to import
   - Google Play Services: google-play-services_lib
   - Other dependencies: UbiNomadLib, Facebook SDK, and foursquare-oauth-library
5. Make sure "Copy project into workspace" is **unchecked**
6. Press "Finish"
7. Congratulations the IDE is ready

B.2.1 Errors

If *UbiNomadLib* has a red "x" in its icon, it may be a build problem. This is solved by following these steps:

1. Right-click on the project and press *Properties*
2. Go to *Android* tab
3. If any library is marked with a red x, remove it and add it again.
4. There should be three libraries included marked with a green checkmark

B.3 Implement In Your Application - Hello UbiNomad

To have a clean setup for a UbiNomad application, this section will go through the steps to set it up.

1. Import an existing Android application
2. Select root directory: *UbiNomadLib*/samples
3. Import *HelloUbiNomad*, and set a name for your application
4. **Check** the "Copy project into workspace" option

5. Press *Finish*
6. A project with your chosen name should be visible in the project explorer. It will also be marked with a red x
7. Right-click on the project and choose *Properties*
8. There should be a red x on the UbiNomadLib project. Remove the project and replace it with UbiNomadLib
9. You should now be able to run the application
10. **(Optional)** Change package name to your liking. If you change the package name, you will need to change it to the same in AndroidManifest
11. **(Optional)** To be able to use Facebook, create an application on developers.facebook.com. Add the application id to *res/values/strings.xml*. Follow Facebooks getting started guide to be able to get key hashes
12. **(Optional)** To be able to use Foursquare, create an application on developers.foursquare.com. Add client_id and client_secret to the constructor when register an oauth key

Appendix C

Extend the Library

There are two ways in which you are able to extend the library: You may add additional place providers, or you may add functionality. This section will describe how to start.

C.1 Additional Place Provider

1. Extend *ExternalProvider*
2. Implement each method the provider supports. If it does not support a function return null
3. Extend *ProviderConnector*
4. Implement the methods in the extended ProviderConnector
5. Add the new provider as new constant in the enum `Providers`, and provide an image.

C.2 Add Functionality

1. Create a new method in the *ExternalProvider* interface
2. Implement the method in each existing Place Provider

C.3 Example: Foursquare

In the validation part of our project, we implemented Foursquare in eight hours. The time it will take to implement a social network depends on how many features the implementation supports.

We started off creating `FoursquareProvider.java` that implements `ExternalProvider.java`. We then had to implement the five functions ExternalProvider consists of. Those functions includes get a list of places near a location, get a single place from a reference, get the connected user, post a status update, and get the Connector. How these methods are implemented is described in section 6.5.

We also needed to implement the `FoursquareConnector.java`, which goal is to manage the connection with the Provider. It tells the application when a connection is started, resumed, or removed.

At last we were able to add the provider to an application by including this line in `MainAcrivity`:

```
ProviderRegister.getInstance().addProvider(Provider.FOURSQUARE, new
    FourSquareProvider("YOUR\_CLIENT\_KEY", "YOUR\_SECRET\_KEY"));
```

Listing C.1: Import Foursquare into application

Appendix D

Terminology

Aggregator place Is a representation of a set of raw places.

Android Studio An integrated development environment for the Android platform created by Google

App Short for application. Is often used when talking about programs made for smartphones like Android

Check-in The action done when you tell an application where you are

Eclipse An integrated development environment

Git A distributed revision control and source code management system

Location A location refers to a specific address or GPS coordinate

Place A place in our context describes a location. For example could someone be at NTNU, which has a location, a category and/or other attributes.

Raw place A raw place is a place gathered from a social network and can be used to check-in at

Smartphone A mobile phone built on a mobile operating system with camera, applications, media player, GPS navigation to make it more advanced than a feature phone

State of the art The highest level of general development achieved at a particular time

Widget Is a relatively simple and easy to use software application made for one or more tasks. In Android this widget term is widely used for the components on the home screen.

Appendix E

Acronyms

API	Application Programming Interface
APP	Application
CRUD	Create, Read, Update and Delete
GUI	Graphical User Interface
GPS	Global Positioning System
HTTP	Hypertext Transfer Protocol
JSON	JavaScript Object Notation
NTNU	Norwegian University of Science and Technology
UbiCollab	Ubiquitous Collaboration
UI	User Interface
XML	Extensible Markup Language

Bibliography

[1] Android activities. http://developer.android.com/reference/android/app/Activity.html. Accessed on: 2014-05-13.

[2] Android architecture. http://www.tutorialspoint.com/android/android_architecture.htm. Accessed on: 2014-05-13.

[3] Android projects. http://developer.android.com/tools/projects/index.html. Accessed on: 2014-06-03.

[4] Android widgets. http://developer.android.com/design/patterns/widgets.html. Accessed on: 2014-06-03.

[5] Difference between location and place. http://www.differencebetween.net/language/words-language/difference-between-location-and-place/. Accessed on: 2014-05-26.

[6] nliteapps. https://play.google.com/store/apps/details?id=com.nliteapps.findmyfriends. Accessed on: 2014-05-08.

[7] Social media fast facts: China. http://socialmediatoday.com/richard-simcott/2213841/social-media-fast-facts-china. Accessed on: 2014-04-18.

[8] Tinder. http://www.gotinder.com/. Accessed on: 2014-05-21.

[9] Ubicollab. http://www.ubicollab.org/about/. Accessed on: 2013-12-11.

[10] Ubishare. https://github.com/UbiCollab/UbiCollabSDK/wiki/UbiShare. Accessed on: 2013-12-09.

[11] Glympse. http://www.glympse.com/, 2012. Accessed on: 2013-11-05.

[12] Open Handsent Alliance. Content providers. http://developer.android.com/guide/topics/providers/content-providers.html. Accessed on: 2013-11-09.

[13] Open Handsent Alliance. Creating a syncadapter. http://developer.android.com/training/sync-adapters/creating-sync-adapter.html. Accessed on: 2013-11-09.

[14] Krzysztof Cwalina. Design guidelines book (the principle of scenario-driven design). http://blogs.msdn.com/b/kcwalina/archive/2005/05/05/scenariodrivendesign.aspx, 2005. Accessed on: 2014-06-03.

[15] Google. Google places api. https://developers.google.com/places/documentation. Accessed on: 2013-11-18.

[16] Android Open Source Project. Android licences. http://source.android.com/source/licenses.html. Accessed on: 2013-11-08.

[17] Samuel Wejéus, Boris Mocialov, Endre Bakken Stovner, Fredrik Bjorland, and Charmaine Geldenhuys. Ubinomad a space manager for ubicollab. 2011.