Graphy: Exploring the Potentials of the Contacts Application

A Thesis Submitted to the
College of Graduate Studies and Research
in Partial Fulfillment of the Requirements
for the degree of Master of Science
in the Department of Computer Science
University of Saskatchewan
Saskatoon

By
Nam Hoang

Permission to Use

In presenting this thesis in partial fulfilment of the requirements for a Postgraduate degree from the University of Saskatchewan, I agree that the Libraries of this University may make it freely available for inspection. I further agree that permission for copying of this thesis in any manner, in whole or in part, for scholarly purposes may be granted by the professor or professors who supervised my thesis work or, in their absence, by the Head of the Department or the Dean of the College in which my thesis work was done. It is understood that any copying or publication or use of this thesis or parts thereof for financial gain shall not be allowed without my written permission. It is also understood that due recognition shall be given to me and to the University of Saskatchewan in any scholarly use which may be made of any material in my thesis.

Requests for permission to copy or to make other use of material in this thesis in whole or part should be addressed to:

Head of the Department of Computer Science
176 Thorvaldson Building
110 Science Place
University of Saskatchewan
Saskatoon, Saskatchewan
Canada
S7N 5C9

Abstract

The number of mobile devices is growing very fast. Smart phones and tablets are step by step, replacing desktops and laptops as the primary method of computing in daily life. According to the 2014 annual report from Cisco [2], there were 406 million new smart phones and 92 million new tablets activated in 2013. Not only are smart devices increasing in popularity but they are becoming more and more powerful. The latest A8X chip which Apple uses for the iPad Air 2 contains 3 billion transistors and has 40% more CPU performance and 2.5 times the graphics performance of its predecessor - the Apple A7 which had been released just a year before. Along with the rapid evolution of mobile devices, the applications on them are undergoing fast transformation. We can see many improvements in traditional applications (messaging, calling...) like multimedia text messages, video calls, voice over IP and so forth. However, the Contacts application has not change much while it has many potentials. In this proposal, we explore the potentials of providing custom information, retaining historical context data in the Contacts application and connecting the contacts together based on their relationships. We implement a prototype of the new Contacts application which utilizes these three aspects then test it among a group of users. Our results are expected to indicate that custom information, historical context data and contact relationships are three important parts which should be included in today's Contacts applications.

Contents

Permission to Use i

Abstract ii

Contents iii

List of Tables v

List of Figures vi

1 Introduction 1
- 1.1 Background . . . 1
- 1.2 Problem Definition . . . 2
- 1.3 Research Goals . . . 4

2 Literature Review and Analysis 6
- 2.1 Contact Information and The Need of Tagging . . . 6
  - 2.1.1 Tagging . . . 9
  - 2.1.2 Historical Contexts Awareness . . . 17
- 2.2 Contact Relationships . . . 24
  - 2.2.1 Social Network Sites: Definition and History . . . 25
  - 2.2.2 Graphy vs. Social Network Sites . . . 26
- 2.3 Cloud-based Contacts Application . . . 30
  - 2.3.1 Overview of Cloud Computing . . . 30
  - 2.3.2 Utilizing Cloud Computing in Contacts Applications . . . 31
- 2.4 Summary . . . 32

3 Design and Implementation 34
- 3.1 System Architecture . . . 34
- 3.2 Mobile Client . . . 34
  - 3.2.1 Technologies . . . 34
  - 3.2.2 Features . . . 35
- 3.3 Backend Server . . . 38
- 3.4 Database Design and Synchronization . . . 39
  - 3.4.1 Database Schema Design to Support Tagging . . . 40
  - 3.4.2 Synchronization Technique . . . 48

4 Experiment and Evaluation - User Study 53
- 4.1 Hypotheses . . . 53
- 4.2 Experimental Setup . . . 53
- 4.3 Results and Discussions . . . 55
  - 4.3.1 Tags and Relationships . . . 56
  - 4.3.2 Contacts Search and Relationship Traversal . . . 61
- 4.4 Summary . . . 65

5 Experiment and Evaluation - System Performance 68
- 5.1 Goals . . . 68
- 5.2 Experimental Setups . . . 68
  - 5.2.1 Overview . . . 68
  - 5.2.2 Clients . . . 68

5.2.3 Server . . . 70
5.2.4 Communication . . . 70
5.3 Scenarios and Results . . . 71
5.4 Conclusion . . . 74

6 Conclusion **76**
6.1 Summary and Expected Contributions . . . 76
6.2 Timeline . . . 76
6.3 Future Work . . . 76

References **78**

List of Tables

2.1 Commonly Used Physical Sensor Types 20
2.2 Summary of literature review 33

3.1 RESTful API 40
3.2 MySQLicious Schema 41
3.3 MySQL Configuration 46
3.4 System Configuration 46

4.1 Total Numbers of Active Contacts, Tags, and Relationships 57
4.2 Tag and Relationship Types 60

5.1 System's Specifications 69
5.2 Time to Completion of The Scenarios (seconds) 72

List of Figures

1.1 Text Messaging Application Evolution . . . 2
1.2 Contacts Applications in iOS 6 and iOS 7 . . . 3
1.3 Contacts Applications in Android 4.4, and Windows Phone 8.0 . . . 5

2.1 ZoneTag users' tagging frequency across their entire Flickr collections (including untagged photos) . . . 10
2.2 Nguyen et. al.'s "Add Tags" UI on A Paper Prototype . . . 15
2.3 Nguyen et. al.'s "Contact Info" UI on A Paper Prototype . . . 16
2.4 Layered Conceptual Framework for Context-aware Systems . . . 19
2.5 Using A Prosthetic Episodic Memory Device . . . 21
2.6 Forget-me-not Implemented on The ParcTab Hardware . . . 23
2.7 Intimate Computing Simple Model . . . 24
2.8 Timeline of the launch dates of various major social network sites and dates when community sites re-launched with social network features . . . 27

3.1 System Architecture . . . 35
3.2 Xamarin Architecture . . . 36
3.3 Model-view-viewmodel Pattern . . . 36
3.4 Main Screen . . . 37
3.5 Contact Profile . . . 38
3.6 Historical Context Tags . . . 39
3.7 Scuttle Schema . . . 42
3.8 Toxi Schema . . . 44
3.9 Intersection Queries with 250 Tag Set . . . 47
3.10 Intersection Queries with 999 Tag Set . . . 48
3.11 Union Queries with 250 Tag Set . . . 49
3.12 Insertion Queries with 250 Tag Set . . . 50
3.13 Tag Schema . . . 51
3.14 Relationship Schema . . . 52

4.1 Graphy's Summary Page . . . 54
4.2 Graphy's Online Form . . . 55
4.3 Number of Active Contacts . . . 58
4.4 Number of Tags . . . 58
4.5 Number of Relationships . . . 58
4.6 Average Tag Weight Distribution among Participants . . . 61
4.7 Average Relationship Weight Distribution among Participants . . . 62
4.8 Average Proportion of Fields in A Contact . . . 63
4.9 Total Numbers of Searches . . . 64
4.10 Average Proportion of Searches . . . 65
4.11 Relationship Navigations Distribution among Participants . . . 66

5.1 Sync Queue Table . . . 69
5.2 Backend Server Implementation . . . 70
5.3 Time to Completion of The Scenarios . . . 73

Chapter 1

Introduction

1.1 Background

As mobile devices continue their rapid run toward global adoption, mobile applications are extensively used in almost every aspect of our daily life. According to new data from Gartner [37], by 2017, mobile applications will be downloaded more than 268 billion times, generating revenue of more than $77 billion and making applications one of the most popular computing tools for users across the globe. Together with the emergence of new applications, the rapid growth of mobile devices in both computing power and popularity leads to valuable renovations in many traditional mobile applications as well. Taking a quick look at the messaging application, for example, we can see a number of improvements (Figure 1.1). Messaging applications today are capable of sending not only plain text messages but also images, videos, and other types of files. Furthermore, we can send messages to any place in the world via the Internet instead of relying on the local telephone companies. Regarding the user interface, in the past we could only read one message at a time but nowadays we can see the whole conversation easily and conveniently thanks to the large and stunning display of smart phones. Not only are messaging applications improved, looking at the calling or note taking features of phones, we can see many enhancements as well.

However, there is one application that has not received much attention in the evolution of mobile phones: the Contacts application. People seem to ignore it although it has many potentials. In the past, the Contacts application was the site of initiating a few types of communications namely voice calls, text messages, and emails. Nowadays, with the emergence of new communication channels such as instant messaging [13, 20], Voice over IP [12, 19], and social networks [4, 8] the Contacts application can play an important role in creating a unified communication management interface. For example, instant messaging applications like Whatsapp [20] and Snapchat [13] have been using the contacts of the devices as the identities for their users in order to eliminate the friends discovering and adding phases of normal chat applications. Some Voice over IP application such as Fring [6] and Viber [19] use this approach as well. Furthermore, in some modern mobile operating systems like Android [1], when users select a contact on their devices they can choose to make a traditional voice call/text message or to use Voice over IP/instant messaging applications. Regarding social networks, we can link multiple social network profiles from Facebook [4], LinkedIn [8] to a phone contact via the similarities in email address and phone number. Beside all the new communication channels, voice

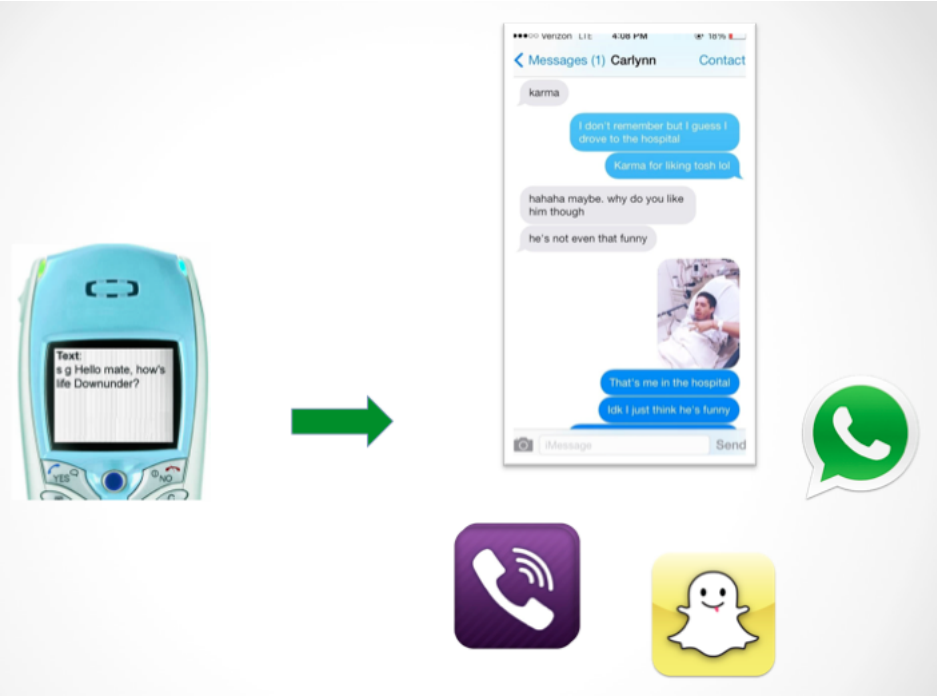

Figure 1.1: Text Messaging Application Evolution

calls and emails still act as the dominant communications in the business world since they are more formal. Therefore, the Contacts application is the central place for managing and widening people's connections.

1.2 Problem Definition

Regardless of playing a key role in users' connections, Contacts applications today have not changed much from their original form. All popular Contacts applications nowadays are merely ordered collections of contacts. As we can see from Figure 1.2 and Figure 1.3, the contact profiles are a little more informative with the additions of profile pictures and some pre-defined fields, and yet, these additions are still limited. There are some issues users still have to face which have not been solved:

1. **Finding the right contact with some particular pieces of information**: T. Nguyen et. al [50] through their study show that the majority of participants sometimes do not know who to call with a piece of information. The problem becomes more severe when it comes to looking up business or service contact information. They are the kind of long-term interaction which are infrequent but long-lasting contacts. The survey reveals that many people do not remember the name they use when creating the contact, hence they often have to browse the entire contacts list to find the number.

2. **Recalling and retaining miscellaneous information of a contact**: When we have hundreds of contacts, remembering who they are and how we met them becomes an extremely difficult task especially for the contacts who we met quickly for business purposes. Whittaker et. al. [57] emphasis

that maintaining knowledge of the contacts is a critical problem. People are being exposed to an unmanageable number of contacts. Consequently, according to the researchers, users must decide: which of these contacts are valuable enough to maintain information about and what kinds of information to retain about the chosen contacts. Furthermore, recording the information is usually laborious and boring.

3. **No support for establishing relationships between contacts**: If we connect the contacts inside one's Contacts application with each other, they will form a network. This network is remarkably similar to a social network like LinkedIn or Facebook. For instance, the profiles in both networks consist of basic information about real people and these profiles can be connected based on their real world relationships. Actually, according to Boyd and Ellison's definition [29], the contact network satisfies all the criteria for a social network. However, this aspect has not yet been explored in modern Contacts applications. Since social networks have been developing incredibly and providing many benefits to theirs users, we believe that not having the capability to establish relationships between the contacts in a Contacts application is a big omission. With the contact relationships, a Contacts application can answer a whole new class of user queries such as "Find all colleagues of contact A", "Find the spouse of contact B". This type of queries is currently impossible to accomplish in Contacts application today.

Figure 1.2: Contacts Applications in iOS 6 and iOS 7

1.3 Research Goals

In addressing the above mentioned issues, this research looks into developing a foundation model for contacts management applications which allows users to:

(a) **find contacts by their miscellaneous information.**

(b) **efficiently retain knowledge of contacts.**

(c) **establish relationships between contacts then traverse and explore the relationships with ease.**

In this regard, our research looks into creating Graphy - a prototype based on modern Contacts applications with improvements. Graphy fulfills the goal (a) by enabling users to create customized information for their contacts through a tags system. With the tags system, Graphy can act as a contacts "search engine" from which users can search for contacts with any particular characteristic. At the same time, the customized information tags also help users retaining various knowledge of their contacts (goal (b)). Besides customized information tags, Graphy automatically adds useful information like the location, the date, the event which the users are attending while they create a new contact to that contact's profile. These additional pieces of information create a context around the contact which assists users in recalling knowledge about it. Regarding goal (c), Graphy allows users to connect their contacts with each other to form an internal network which we call "reversed social network". We will explain details of this network and why we call it like that in Chapter 2. Furthermore, the emergence of Cloud Computing provides modern Contacts applications with the elegant capability of backup and sync data between multiple devices. Therefore, we want to ensure that our prototype is capable of exchanging data with the cloud using state-of-the-art technologies. Since all popular modern Contacts applications are closed source with proprietary software, we decide to develop our own communication and synchronization solution utilizing the trending RESTful communication technology. Our solution aim to make it easy for users to work offline through a local database while maintaining a central cloud database to backup and synchronize data to other devices when needed.

In short, the objectives of our prototype are as follows:

- Operating like a modern Contacts application as well as providing additional improved features.
- Allowing users to create customized information tags.
- Automatically adding contextual information like the creation date and the creation location of a contact.
    - ⋆ The customized information tags and the contextual information should be searchable with rapid responses.

- Allowing users to establish relationships between contacts in a bi-directional manner.
  - ⋆ The relationships should be easy to traverse and explore.
- Capable of backing up and synchronizing data to a server using RESTful communication.

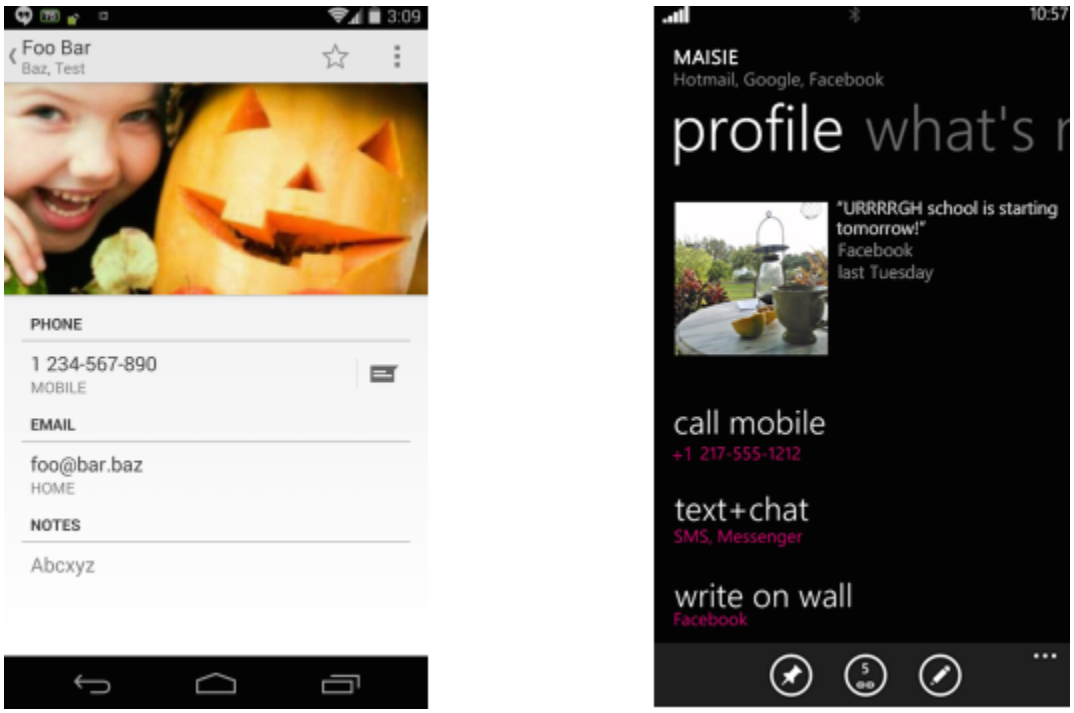

Figure 1.3: Contacts Applications in Android 4.4, and Windows Phone 8.0

Chapter 2

Literature Review and Analysis

In this chapter, we review the state of the art and the related work that has influenced our approach. At the end of each section/subsection comes a short summary with analysis or comparison. First, section 2.1 looks into the design drivers a modern Contacts application should follow. It goes deeper into investigating how to apply these design drivers using tags and historical context awareness in subsection 2.1.1 and 2.1.2. Subsection 2.1.1 reviews users motivations and the use of tags in different existing systems, the implications for designing a tagging system in general, and how tags should be applied on mobile devices. Subsection 2.1.2 studies the design principles of context-aware systems then comprehensively examnines Forget-me-not, a context aware prototype of a memory aid device. Second, in section 2.2 we look into various modern social network sites then define and compare our "reversed social network" with them. In section 2.3, we discuss the fundamentals of cloud computing and the benefits of utilizing cloud computing in Contacts applications. Finally section 2.4 summarizes all the aspects having been reviewed above.

2.1 Contact Information and The Need of Tagging

Most of our daily communication activities involve managing contacts information. Through a field study from Whittaker et al. [57] the value of contact information is strongly affirmed. Mary, a participant in the field study stated that her personal contacts list was a resource which pervaded all of her work:

"I cannot work today unless I have some source of contact information, some organized source so I can actually actively search for people. I use this list all the time just to browse it to find people when I need somebody to do a particular task."[57]

Understanding the importance of the contacts list, Jung and other researchers at Nokia [43] introduce eight design drivers a Contacts application should follow:

1. **Efficiency of accessing contacts**: Apparently, speed and ease of accessing and creating new contact information are key factors of the Contacts application. They should be the first priority in the design process.

2. **Differentiating important contacts**: Interviewed participants in Jung et. al. study expressed their demands in differentiating special contacts from others for both emotional and practical goals. As the size of mobile contacts list grows, many participants manually changed the order of appearance of

some contacts in the list (for example, by adding a number "1" in the name field) or attaching a visual mark to make the contacts stand out of the list (for example, adding symbols like "*" or a heart "<3" to indicate his/her significant other). However, the nature of relationship often changes over time so manually altering contacts as shown above is obviously tedious and not ideal. Therefore, on a practical level, the researchers recommend using a dynamic list which can be automatically generated based on some conditions like frequency and recency of communication to differentiate important contacts.

3. **Customization and personalization**: Jung et. al. claim that the Contacts application is a highly personal application which accommodates different user preferences and lifestyles. Hence, it is essential to have a model to understand the range and patterns of user preferences which can be interpreted into user preference settings. The researchers then propose the feature that allows personalization of content on various levels such as adding a picture, an icon, a note to a contact profile.

4. **Contact as repository of personal information**: Nokia researchers show that contacts list is not only about other people. Their interview revealed that users store personal information which is unrelated to communication management in their Contacts application. This information could be passwords or account numbers or other data that may potentially be retrieved more than once. Some advanced users developed some special ways to "encrypt" their private data to make it secure (for example, naming the credit card PIN number as "Jonny" in the contacts list). In addition to users' own personal information, through their study, the researchers found it is necessary to promote users storing others' digital identities in the Contacts application.

5. **Contact as social piggy bank**: People often consider adding other information which is related to the social relationship to the contact profile if the application has that functionality. Active users in Jung's study wished to have the ability to add further information about the person such as birthday, social network, communication history to provide or strengthen the context of the relationship, especially when the user has a strong social connection with the person. Therefore, it is essential to design a contact profile that can accommodate both the people-centric and the information-centric types.

6. **Assistance to social management**: The study revealed different areas where people will benefit from making use of past communication patterns (for example, storing the date of when the contact profile was created, which may mark the moment when the user first met that person). In contrast with the "Contact as social piggy bank", this design driver helps users who do not proactively add extra information since most data of this kind could be accumulated and stored automatically.

7. **Flexibility in organization**: Jung et. al. stated that their research revealed various potential reasons why people may use hierarchical or grouping organization in the contact profile. One of the reasons was to assign common settings like special ringing tone to a group of contacts. Another reason was to assist faster access and reduce the visual clutter when there were many contacts with the same names.

8. **Accessing external contact information**: With the powerful mobility of mobile phones, sometimes accessing the basic information of a foreign area is necessary in an unexpected circumstance (for example, taxi, hospital, directory service). Therefore, it is desirable to have access to the locally related data. The researches suggested that instant access to external contact information database has a big potential in improving the Contacts application.

The design drivers listed above were published in 2008, one year after the first iPhone flipped the world upside down by introducing a whole new concept of touch-screen smart devices. Over the years, these drivers have been proven to be applicable and implemented completely or partially by manufacturers. Taking a look back into the past, the first Contacts application on a mobile phone was simply a dictionary of names and numbers. A contact profile used to consist of only the person' s name and his/her phone number. Nowadays, a contact profile in major platforms (iOS, Android, Windows Phone) is not only the person's name and phone number but also their social profile. This social profile is made up of the person's picture, name, phone numbers, and a group of miscellaneous fields such as company, job title, address, birthday, emails, notes, and so on. Furthermore, the Contacts application provides users with many extra functionalities such as logging communication history, marking favorite contacts, grouping contacts. We can see that almost all the design drivers are implemented in modern Contacts application except for design drivers number 4, 8, 3 and 6. As a side note, we do not sort the design drivers by numerical order here but we sort it by its content explained below:

- **Design driver number 4 - Contact as repository of personal information**: At the time Jung et. al. proposed the design drivers, smart phones were not yet powerful. Therefore, it is understandable that some people utilized the Contacts application to store their personal information like account number, usernames, passwords. However, with the capabilities of modern mobile phones, users have many choices to accomplish that task. For example, it is much better to save private information using a password manager application since it is more convenient and the data is safely protected using modern encryption algorithms. For other kind of information, users can choose to store it in a note taking application or a task manager which are available on all mobile platforms. In summary, this design driver has become inappropriate in today technologies.

- **Design driver number 8 - Accessing external contact information**: Having the ability to retrieve external and location-related information directly from the Contacts application is desirable. However, the enormous power of today search engines together with the rising of mobile intelligent personal assistant like Siri (iOS), Google Now (Android), Cortana (Windows Phone) has made accessing external information incredibly easy. Users can just verbally command the intelligent personal assistant or type a few keywords in an on-screen widgets then the relevant information will be retrieved instantly from a search engine. Therefore, the majority of users has developed a habit of looking up for external information via search engine, hence using the Contacts application for this task becomes redundant

and sometimes slow. To conclude, accessing external contact information from the Contacts application has turned out to be just a "nice to have" feature without much potential thus not being implemented in most modern Contacts applications.

- **Design driver number 3 - Customization and personalization**: Unlike the two obsolete design drivers we have just analyzed, design drivers number 3 and 6 are actually implemented in almost all Contacts applications today. Design driver number 3 is about customizing and personalizing the contacts. In modern Contacts applications, this driver exists in the form of the picture in the contact profile and a group of extra fields like websites, phonetic name, notes. However, all those fields are pre-defined by the application. To the best of our knowledge, all Contacts applications lack the ability for adding customizable fields. In other words, we consider the pre-defined fields only fulfill the "Personalization" half of design driver number 3 while missing the first half - "Customization". In 2010, Trung V. Nguyen et. al. [50] conducted a survey with the participation of 87 people in Korea which revealed that Contacts application users need various extra customizable information related to a contact. They then pointed out that memos or notes are not sufficient for this purpose, and proposed the use of tags. We are totally in unison with this idea of using tags for customization and we will discuss more in details of this topic in section 2.1.1.

- **Design driver number 6 - Assistance to social management**: Assistance to social management is another limitedly implemented design driver. From our analysis, all major Contacts applications today only record the calls and text messages log of a contact while there are many more potential past communication patterns. For example, the date and the location a contact was created may tell a lot about how the user meets that person. The survey from Trung V. Nguyen et. al. [50] shows that many people struggle with remembering who the contacts are and who they should call for a piece of information. In these situations the past communication patterns can be very helpful. We will analyse this problem more in section 2.1.2

2.1.1 Tagging

Tagging is using keywords to add metadata to content [38]. It has become very popular in applications and services nowadays [31, 38, 47]. Many types of data are currently annotated by tags which includes bookmarks, images, videos, articles, blogs and so on. There are different services providing tagging functions to data such as Pocket, Pinboard, Delicious, Flickr, Youtube. Tagging is effectively enhancing users experience in organizing and sharing large amounts of information in those services.

In order to understand tagging, Ames and other researchers at Yahoo and Stanford University conducted a large study on more than 500 people [25]. The study focused on an images tagging system combined of Flickr and ZoneTag. Flickr is an image hosting web services. It is a popular website having one of the largest online communities. Flickr provides annotation of images in the form of tags. A tag in Flickr is

an unstructured textual label and mainly assigned by the user who owns that image. ZoneTag is a mobile application developed by the researchers. It helps users upload a newly captured photo from their mobile phones directly to Flickr in just a few clicks. More importantly, the users can type in or select tags for that photo. This feature pre-fetches a number of tags from the ZoneTag server based on the context then suggests them to users for easy selection. ZoneTag was deployed as a public prototype for around a year with more than 500 users and 45,000 uploaded photos. Ames et. al. then carried out a study on 172 users. During the deployment process, Ames collected their data about the usage of the system. The data was used to analyze tagging patterns and activity. Figure 2.1 demonstrates the percentage of users regarding their average number of tags per photo. For instance, the first bar on the left indicates that 25% of the ZoneTag users have less than 0.5 tag per photo. Notably, this number does not count the automatically added tags. As we can see from the chart, around 61% of the users added at least one tag for each photo on average. Surprisingly, over one fifth of the users had more than 3 tags per photo which reveals a great potential of tags. All the tags of the system come from ZoneTag mobile application as well as Flickr's official website. However, the majority of tags were from ZoneTag with over two thirds of the tagged photos were added by users with their mobile devices.

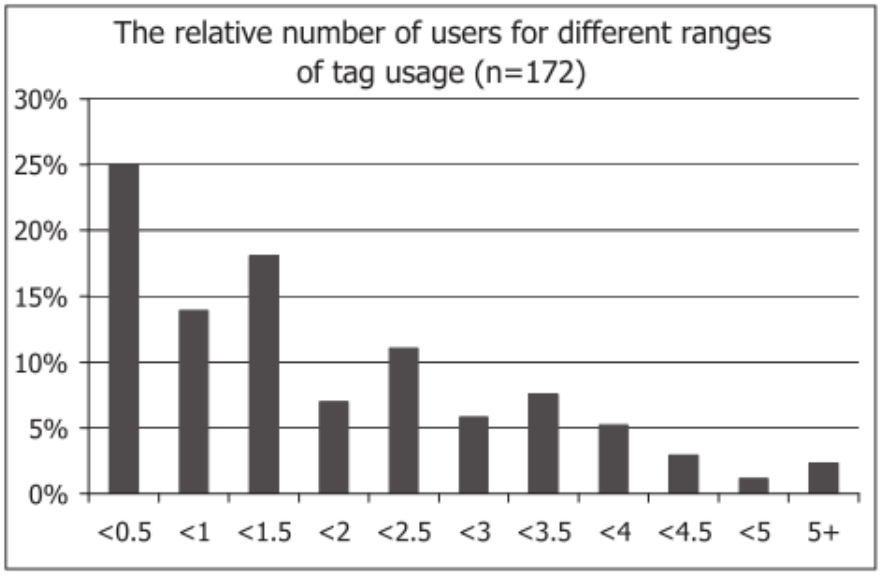

Figure 2.1: ZoneTag users' tagging frequency across their entire Flickr collections (including untagged photos)

In addition to analyzing collected data from users, Ames et. al. conducted an in-depth, semi-structured interview with 13 participants including some users who had taken the most photos. Their ages ranged from 25 to 45 with four of them being female. The tagging frequency of each participant covered from zero to more than five tags per image. The interview revealed many type of motivations and uses of tags. Almost all the participants had more than one motivations for tagging. For example, an interviewee tagged her photos in order to retrieve them later as well as to provide contextual information of the images for herself and

her friends. The researchers established a taxonomy for the motivations which included 2 categories and 4 groups:

- *Self/Organization: Search and Retrieval*

  This is the traditional motivation of tagging. Users who have this motivation occasionally make comments like "I am an organized person" or "I like order". However, these comments are often followed by admission for not being consistent in the tags. Two of the participants say that they tag especially to later retrieve their photos and two others say they tag for personal organization purpose. Below is the quote from one of these participants:

  "Mostly I use tags if I go back on Flickr if I want to find all the pictures of one thing. If I tagged ahead of time I can go back and get all my pictures of my children. ... I've made separated tags of my child's preschool or playgroup so that if I want to share pictures with more than just family I can go back and find everything from that one tag. ... Mostly it's for my own organization at this point."

  Analysis: This traditional motivation is totally appropriate with a Contacts application. In order to improve users' consistency we should provide a good tag suggestion/recommendation mechanism.

- *Self/Communication: Memory and Context*

  Users sometimes add tags to give context to a photo like the names of the people in it or the location that photo was taken. This behavior improves future recall of the situation the photo illustrates. However, it is surprising that not many users were motivated in this way when they tagged their photos. The quote below demonstrates a user's perspective:

  "If I have the time, the neighborhood, or the event, I have enough information to look at my own collection and know where this came from. I don't have the bandwidth to tag for the benefit of the Flickr system:.. I want at least one hook of association in there that can help me reconstruct what I was thinking. I don't have time to put all the hooks in but I can put one in."

  Analysis: According to the study, there are only a few people having this motivation. From our point of view, there are two problems here. First, context information is ambient data so having it together with the primary data (photos, contacts) is always good so long as it does not distract the viewing process. However, adding ambient data is time consuming and often not appealing, hence not many people spend time doing it. Therefore, this kind of data should be added automatically by the application itself and provide them when needed. Second, a photo provides a lot of ambient information like the people in it, where it was taken. Just by looking at a photo, viewers can understand much about the situation surround that photo. On the other hand, a contact in the Contacts application does not tell a lot about itself. Thus, having ambient information in a contact can help user to recall things about that contact like where and how they met. It is really useful in case there are more than two contacts having the same name. In conclusion, we think this motivation is appropriate with a Contacts

application and needs to be implemented properly. Another thing we wish to emphasize is sometimes it is not clear to differentiate a tag is the first type (Search and Retrieval) or the second one (Memory and Context). Therefore, to be consistent, we consider the second type (Memory and Context) always ambient data which is automatically added by the application.

- *Social/Organization: Public Search and Photo Pools*

  This group represents the users' motivation for making their photographs searchable by other people. The researchers point out that pictures are often taken to enhance and document mutual experience or to share it with friends, family, and even the public. The tags can make the photos be easily found by people the users want to share with or anybody who is interested in them. While some users tag for their friends and family, other users tag for the general public. Below is the quote about this aspect:

  "Most friends view my photos, but as I grow my collection, I am getting more public views. I've noticed that if I take and tag pictures of cute female friends, views go up. ... There's a satisfaction that 50 people have viewed my photos. I know that tagging can connect my photos to activities, and get more interest. ... I got more liberal about using suggested tags lately, so I will add multiple tags to make it easier for people to find my photos."

  As more users tag their images, appealing behaviors arise between groups of friends. Two of the participants said that they coordinated tags with others to assist later retrieval. This is a form of an ad-hoc, distributed photo "pool". One participant did this with his friends in many situations like company meetings, parties, classrooms or hikes with friends. Another participant attended a race in San Francisco then he utilized the tags that others were using to tag his own pictures as well as finding other photos of the event. The quotes reflecting these behaviors are listed as follows:

  "I'm at an event and there's a convergence on a specific tag, then I'll tag because it's for the good of the group. ... It's a nice way to build live streams and collections of photos. ... A classmate suggested we tagged everything specifically so we can find it, which is actually really useful."

  "If I'm out with friends they might suggest tags. ..."

  Analysis: This motivation is not exactly relevant to a Contacts application since the contact information is generally private and often not shared with the general public. However, ad-hoc sharing is an interesting aspect our Contacts application can learn from Flickr and ZoneTag. Sharing a contact with its tags and additional contextual information might be helpful to the receiver since he/she will probably have sufficient contextual knowledge to understand the tags. Nevertheless, this feature should be designed with care because tags often contain a great amount of personal information which the author may wants to keep for only himself. That is why, the ability to select which tags to be shared is essential in this feature.

- *Social/Communication: Context and Signaling*

The last group is tagging to communicate contextual data to other users. In almost all cases, users added these contextual tags for friends and family. Contextual tags for known people often have little meaning for the public. Actually, in order to improve privacy, some users confuse their tags on purpose to make it impossible for the general public to understand the tags. Additionally, two participants described tagging as "a chain reaction", when somebody took a photo and tag, others also took out their phone. One participant describe this perspective as follows:

"I tag and I don't have to explain myself - my friends don't have to ask me a billion questions such as 'where did you take this photo, why are you showing me this photo, who is this person in this photo'... I can give them the basic story."

Analysis: This is another motivation with little relevance to our Contacts application. Since the contact data is not exposed to the public, we do not have to worry much about privacy issues. However, the "chain reaction" is a good sign from which we can believe that tagging in a Contacts application can be widespread.

From the interview, Ames et. al. also analyzed the best way to design the tags suggestion feature. First of all, the researchers decided to make interactions between users and the main activity as burden-free as possible. Particularly, ZoneTag was designed to do its main job in only 2 clicks and it did not require users to add or select tags right away. The reason behind not making adding tags mandatory is that many users from the interview found it difficult to add or even select tags on the phone in some situations like driving or socializing, furthermore, even participants who added many tags still did not want to be interrupted all the time. Secondly, many participants liked having previously-used tags showed up and they also used the auto-completion feature widely. However, the researchers also found out that tags sometimes confuse the users when they share them with each other. One participant complaint about an unfamiliar tag on her photo after using the tags sharing feature in a conference. Nevertheless, from our point of view, sharing photos and sharing contacts are very different since we often share a much smaller set of contacts with just one or two people so the confusion should not be a problematic issue. Lastly, Ames et. al. discovered that tags suggestions served a larger purpose than just assisting in tag entry. Some users had developed the habit of browsing the suggested tags list to add all the tags which were relevant even if they did not try to add them in the first place. One participant commented that he often scrolls down the list and picks available tags because they are displayed. Moreover, even when not being selected, the suggested tags still encourage some participants to add their own new tags and give them direction to the types of tags they may use (for example, a suggested tag about a neighborhood may inspire the users to add tags about other neighborhoods as well). In summary, tags suggestions have a significant impact on users' tagging activity, however the option to bypass adding tags in some circumstances is important for the usability of the whole system.

In the end of the study, after carefully analyzing users' data as well as the interview, Ames et. al. summarized their findings into a list of implications for the design of tagging systems in general [25]:

- Make the annotation ubiquitous and multi-functional.
- Make it as easy as possible when the data (photos, contacts...) is captured.
- Do not force annotation at the point of information capture.
- For systems that have both mobile and desktop or web-based components, annotation should be enabled in both settings.
- Relevant suggested tags can inspire tagging and direct users to possible tags.

We have just examined principles of a general tagging system, we will now investigate how tags should be applied on the mobile phone's Contacts application. In 2010, Nguyen et. al. conducted a user study [50] to discover users' dissatisfactions with contacts management. The study consisted of a large survey which incorporated multiple choice and open-ended questions so participants could point out changes they wanted to be made in their current Contacts application. The survey focused on finding the issues with storing, searching and managing the contacts. It was conducted through an online form over the course of one week with the participation of 87 people. There were 42 male and 45 female participants and their ages ranged from 15 to 50. First, the survey asked participants on the way they organize their contacts. 46 participants said that they use groups for organization while 42 participants did not. However, almost everyone thought that classify the contacts using groups was inconvenient. 55 participants did not know which group they should put a contact into, 12 had difficulty finding appropriate names for their groups, and 11 people claimed that they could not search for contacts using group names. The issues were revealed more clearly through the open-ended questions from which 5 people wanted to put a contact into various different groups while 4 other participants wanted to have hierarchical groups. From these answers, Nguyen et. al. concluded that people use groups as a way to label multiple pieces of information of a contact which may help them recall and fetch the contact when needed. Furthermore, the researchers also addressed the problem when people forget about contacts they have created earlier. 45 out of 87 users said that they did not know who they should call for a piece of information. The problem became more serious when it passed on to business service contact information. Business service such as restaurants, childcare, repair and maintenance, transportation are actually really important. They are the kind of long-term interaction which are infrequent but long-lasting contacts. The survey revealed that 47.5 percent of the participants call this kind of service contacts at least once a month and more than three quarters of all participants call more than once every three months. Despite the importance of these contacts, participants often struggled to look for service numbers in their mobile Contacts application. Among 87 people, 61 did not remember whether they had stored the demanding contact or not, 40 did not recall the name they used when created the contact, and 21 had to look through the entire contacts list to find the number. This fact undoubtedly indicated the need of a system which can assist users in finding the right contact when they have some particular pieces of information about the person/service.

The study also disclosed several solutions which users created manually by themselves in an attempt to tackle those problems. 7 users used memos or notes to store extra information about their contacts while some others put the information into the name entry of contacts just like the way participants in Whittaker's study did which we have examined in the beginning of section 2.1. From this revelation, Nguyen et. al. concluded that users need an efficient mechanism for storing and retrieving extra information of a contact. Therefore, the researchers proposed the use of tags which were convenient for adding and searching resources. Resources could be classified by numerous tags rather than using a directory or a single branch of hierarchy [49]. As a result, users could "tag" a contact into multiple groups or even made the tags hierarchical based on their needs. Regarding retrieval, multiple tags could be used simultaneously in a query for a specific contact so the users did not have to remember the person/service name.

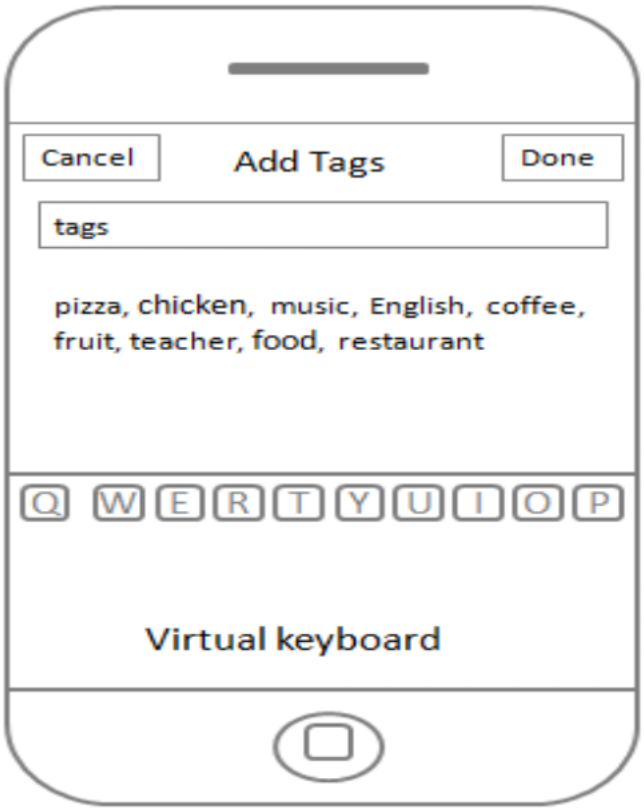

Figure 2.2: Nguyen et. al.'s "Add Tags" UI on A Paper Prototype

After conducting the survey, Nguyen et. al. created a paper prototype of a mobile phone Contacts application to verify their findings. The paper prototype was used in various user study scenarios. At the start of each scenario, the researchers explained to the users new features in the Contacts application. After that participants were requested to perform tasks by interacting with the paper prototype. A researcher acted as the application by responding to users' interactions with other user interfaces also on papers. Figures 2.2 and 2.3 are two example user interfaces of the paper prototype. After finishing a task, the participant filled in the task assessment form and answered some questions from the researchers. All participants thought the

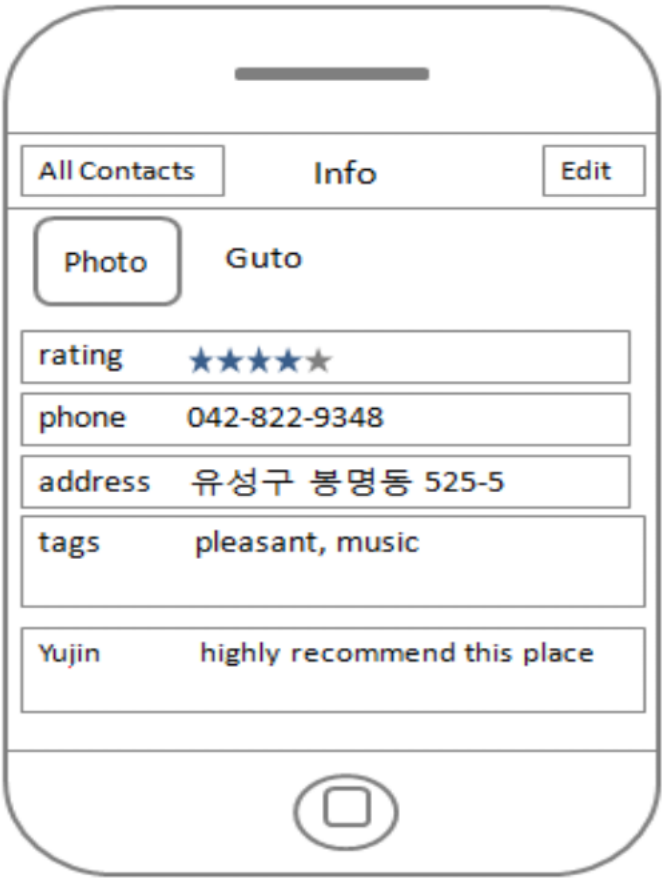

Figure 2.3: Nguyen et. al.'s "Contact Info" UI on A Paper Prototype

tasks were realistic and they had come across many times in real lives. They also commented that the tasks covered all the information they needed when using the Contacts application. In general, the participants had positive feedbacks about the prototype. For instance, a user liked the tagging feature and actively used it a lot through out the scenarios. Furthermore, most of the users only used the name of the service as search keyword once every 20 scenarios, instead they often looked up the contact by using information from the scenarios which were remarkable, relevant, and personal. This was a definitely good sign which indicated the capabilities of tags in directing users to the right contacts from miscellaneous information. On the other hand, some participants admitted that they are sometimes too lazy to enter text on their mobile phones even though they knew that some information about a contact was absolutely necessary. This issue again pointed out the importance of tags suggestion which we have already covered in previous sections.

Summary: To sum up, we have comprehensively investigated a study about tagging in general and another study about tagging in a mobile Contacts application. Both studies suggest that tagging is an essential aspect which could help users organize their data. From Flickr/ZoneTag, we understand the motivations behind tagging, and the recommendations from the researchers will direct us when we build our own system. From the study in Korea, tagging in a Contacts application is once more affirmed to be necessary, and the

paper prototype is really helpful for us in designing our application's user interfaces.

2.1.2 Historical Contexts Awareness

As we have discussed at the beginning of this chapter, the design driver number 6 - Assistance to social management is not fully implemented in Contacts applications nowadays. In order to utilize this design driver, we propose the use of past activities in the form of historical contexts. The contexts here can be many aspects around a contact which happened in the past (this is the reason why we use the term "historical contexts" to distinguish them from the "current context" which is often used to determine the status of the users). For example, the location and the date/time when a contact is created are useful historical contexts which can tell a lot about the situation when the user met that contact, and the events, meetings happened around that time, other contacts added just before and after that contact are meaningful pieces of information as well. Furthermore, the communication history between the user and a contact via emails, call log, and social network interactions can also be considered valuable historical contexts. With the assistance of historical contexts, users can use their Contacts application to recall memory about a contact when needed. Looking back at our analysis of Ames' tagging motivations in section 2.1.1, memory and context is the second motivation of tags. Therefore, we decide to make historical contexts a kind of tag in our system. This kind of tag will be automatically included into a contact profile of the Contacts application along with users-created tags. To understand more about historical context, in this section we will first investigate the definition of context, design principles for context-aware systems, then we will examine "Forget-me-not" an intimate computing system which use context to support human memory.

According to the survey from Baldauf [26], the term "context-aware" first appeared in Schilit and Theimer's work in 1994 [53] in which the authors defined context as locations and identities of nearby people and objects. In 1998, Ryan [52] referred to context as the user's identity, surrounding environment, location, and time. From another angle, Dey [33] not only described context as location, time, and user's environment but also as user's emotional state, focus of attention. However, in the opinion of Baldauf, the most accurate definition of context was given by Dey and Abowd [23] as "any information that can be used to characterize the situation of entities (i.e., whether a person, place or object) that are considered relevant to the interaction between a user and an application, including the user and the application themselves." Regarding context classification, a common way to organize context is based on its dimensions. Prekop et. al. [51] and Gustavsen [40] classified the dimensions into external and internal. The external dimension is the physical context which can be calculated by hardware like location, time, and movement. The internal dimension is the logical context which can be measured only by the users themselves or captured by monitoring users' interactions. Some examples of internal context are the users' tasks, work context, or the users' emotional states. All most all context-aware systems utilize external context since they give valuable information while being easy to measure using modern hardware sensors. There are also some systems making use of logical context like Watson [30] and IntelliZap [36] that assist users by presenting relevant information based on information

extracted from users' opened web pages, and documents.

When designing a context-aware system, the method to capture context information is crucial since it greatly influences the architecture of the system. Chen et. al. [32] demonstrated three different ways to capture context information.

- *Direct sensor access*: This method is widely implemented in hardware devices with built-in sensors. The software collects the needed contextual data directly from the sensors, hence, no extra layer for gaining and processing sensor data is required. However, drivers for those sensors are attached to the application layer forming a tightly coupled system which is not flexible. As a result, this approach is not suitable for distributed systems because of its direct access nature.

- *Middleware infrastructure*: This method divides context-aware systems into layers. The main purpose of this approach is to hide low level sensing details. In comparison with the first method, this layered architecture approach is more flexible and extensible. The application layer is separated from the sensors so it is easy to reuse hardware dependent sensing code. However, the architecture is more complicated compared with direct sensor access.

- *Context server*: This method expands the middleware approach by introducing a remote server. This server aggregates sensor data then provides access to authorized clients. The first advantage of this technique is reusing sensor and introduce an administration layer for the contextual data. The second advantage is saving the clients from resource intensive operations since almost all client devices in a context-aware systems are mobile devices with limited computing power, memory, and energy. On the other hand, this approach requires a complex communication protocol between clients and the server, and the network performance can sometimes become the bottleneck of the system.

Regardless of what method a system chooses to capture contextual data, logically separating context detection and context consumption is essential in enhancing extensibility and reusability of the system. Figure 2.4 illustrates the layered logical architecture which was described by Ailisto et. al. in 2002 [24].

The first layer is called the "Sensors" layer. The term sensors here not only means physical hardware sensors but also means any kind of data source that supplies valuable contextual information. In accordance with Indulska and Sutton [42], sensors can be categorized into three different groups based on the way they capture data.

- *Physical Sensors*: This is the most common type of sensors. They are the actual hardware sensors that can capture many kinds of physical information like longitude, lattitude, temperature.... Physical sensors are very popular today and can be found in almost all smart devices or mobile phones. Table 2.1 lists some common physical sensors.

- *Virtual Sensors*: This type of sensors get contextual information from software or services. For instance, an application can track users' location not only by using GPS data but also by reading their electronic

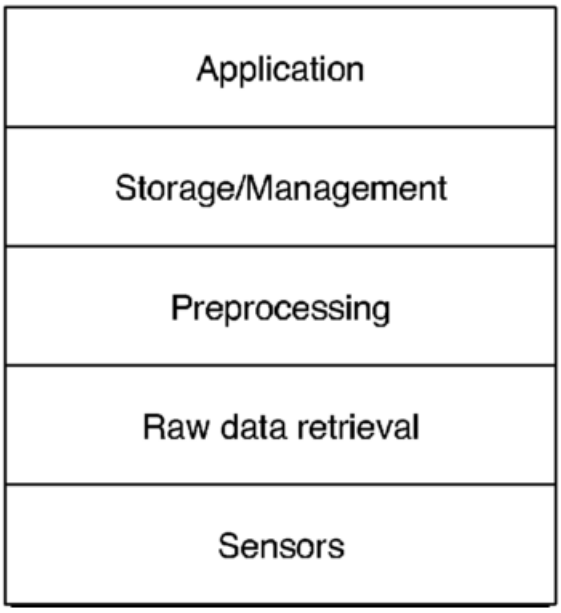

Figure 2.4: Layered Conceptual Framework for Context-aware Systems

calendar, emails.

- *Logical Sensors*: Logical sensors take advantage of multiple data sources by incorporating physical and virtual sensors with supplementary information from databases or other sources to deduce more complex information. For instance, logical sensors can be used to track employees' current location by examining user logins on company's machines and combine those pieces of information with a database storing machines' locations.

The second layer, *Raw data retrieval*, is for retrieving raw contextual data. It utilizes drivers on physical sensors or APIs on virtual and logical sensors. This layer provides abstract functions for the upper layer in order to make it easy for accessing the low-level hardware layer. Furthermore, by abstracting the access interfaces, it is possible to replace the hardware sensors module with another one without changing the upper layer's program. For example, replacing a RFID sensor with a GPS one can be done seamlessly.

The third layer is not commonly implemented in context-aware systems. It is used for interpreting context information which is sometimes important when the raw data is too harsh. In some systems, the technical data from the sensors is not directly useful for the high-level software application. This is when the preprocessing layer is brought into play, it transforms the data into higher abstraction information by using extraction and quantization operations. Moreover, in systems which have more than one context information source, the data of all sources can be aggregated in this layer before delivering to the next layer. The aggregation process is often important since in many scenarios the data from a single sensor does not make sense or is inaccurate while the combined data from multiple sources is valuable and much more precise.

The forth layer is called *Storage and management*. It organizes and store the collected data then provides

Type of Context	*Sensors*
Light	Photodiodes, color sensors, IR and UV sensors, etc.
Visual	Cameras
Audio	Microphones
Motion, acceleration	Mercury switches, angular sensors, accelerometers, motion detectors, magnetic fields
Location	Outdoor: Global Positioning System (GPS), Global System for Mobile Communications (GSM); Indoor: Active Badge system, etc.
Touch	Touch sensors in mobile devices
Temperature	Thermometers
Physical attributes	Biosensors for measuring skin resistance, blood pressure, etc.

Table 2.1: Commonly Used Physical Sensor Types

them through interfaces. The applications/clients can receive the data synchronously or asynchronously.

They highest layer in the stack is the *Application* layer. All the business logic that makes use of the stored contextual information is carried out here.

The next important thing we need to know when designing a context-aware system is the context models. The context models represents the machine readable form of the contextual data. It is challenging to develop a scalable, adaptable, and usable model which can cover a broad range of potential contexts. In their work "A Context Modeling Survey" [54], Strang and Linnhoff-Popien listed the most common context modeling methods. These methods were categorized based on the data structure representing context data.

- *Key-value model*: This is the most simple data structure which can be used for modeling contextual information. Although this model is simple, it is very effective and is used in many systems. According to Strang and Linnhoff-Popien, it is especially used a lot in service frameworks in which the key-value pairs represent the capabilities of a service. As a result, service discovery can be applied by running matching algorithms on the key-value pairs.

- *Markup scheme model*: This kind of model makes use of markup tags with attributes and content to construct a hierarchical data structure.

- *Graphical model*: Graphical model mostly utilizes the Unified Modeling Language (UML) which can be used to model context data. Some graphical model also extends the Object-Role Modeling (ORM) to represent context.

- *Object oriented model*: This approach can leverage the power of object oriented techniques like inheritance, encapsulation, etc. Various context types can be represented by different objects.

- *Logic based model*: This model has a high degree of formality. Context models are defined by facts, expressions, and rules. After that, a logic system will manage those facts, expressions, and rules as well as allowing addition, update or removal of each element. Reasoning processes are used to develop new facts based on existing rules in the system.

- *Ontology based model*: Ontologies are considered a potential way to model context data since they have a high level of expressiveness while maintaining the possibilities to apply ontology reasoning methods.

After investigating architecture and design principles of context-aware systems, we will now thoroughly examine Forget-me-not, a context-aware program used for Intimate Computing. Forget-me-not [44] is a project attempting to find a method to assist human memory using mobile and ubiquitous computing. It aims to solve the problem of growing information load in daily life.

Figure 2.5: Using A Prosthetic Episodic Memory Device

In order to develop Forget-me-not, the researchers introduced a new computing model called "Intimate Computing". It was based on the definition of Ubiquitous Computing by Weiser [56], it consisted of tiny Personal Digital Assistants (PDAs) with wireless communication capability such as cell phones, laptops,

wearable devices. Those devices always escorted the users so they can be tailored to their own preferences. Moreover, since they were involved in many of users' daily activities, the devices became intimate with them. The more intimacy the devices had, the more valuable they were. Furthermore, it is notable that Intimate Computing provided the computing devices with the users' real world contexts.

Utilizing Intimate Computing, the researchers attempted to solve the problem of forgetting one's own information. Since the PDAs had access to users' contexts, they could use the context as a useful key to index information automatically. According to the authors, a detail of a past event like the name of a document was probably hard for a user to remember. However, the context of that event could be easier to recall such as the person who gave the document to the user, the place when it happened, the task the user was doing. Psychology researchers also developed theories about this kind of physical contexts. They called it episodic or autobiographical memory. The psychologists discovered that people instinctively arrange memories of past events into episodes, and then the location of the episode, the people around, the activities that had happened before, during, and after the episode were solid clues for recall. Furthermore, a study by Eldridge et. al. [34] even led us to believe that we could make a prosthetic episodic memory device. The device was called a *memory prosthesis* which followed users and captured critical information and context from their lives, then it could organize this information into a structure which mimicked the episodic memories of human beings. As a result, people could retrieve details of their fading memories by looking up the episodes which were accumulated in the storage of their prosthetic memory devices (Figure 2.5). In other words, users could use small, easy to remember things about a context to bring back the details that they had forgotten.

Forget-me-not was Lamming et. al.'s first attempt [44] in creating a functioning prototype of a prosthetic episodic memory device. The first aspect of the device which the researchers tackled was user interfaces. They tried to design the interfaces to be as easy to use and intuitive as possible since it should be easier to remember how to operate Forget-me-not than to remember past events. Figure 2.6 shows the Forget-me-not device developed in 1994. The software was implemented on a ParcTab - a portable device built in the Computer Science Laboratory at Xerox PARC.

By accompanying users, the ParcTab accumulated data about different users' activities then it arranged these data into a personal biography. In the simplest prototype, Forget-me-not required users to provide a list of devices where data could be gathered. When a user interacted with a device on the list, ParcTab automatically collected the device's name and location. The operations carried out by the user on that device were also collected together with a timestamp. When two people both wearing ParcTabs met each other, their devices would start exchanging information in regard to their preset privacy settings. Figure 2.7 illustrates this model. According to the authors, the Forget-me-not prototype was deployed for a few months and to some extent confirmed the applicability of the Intimate Computing model.

Summary: To sum up, in this section we have decided to include historical contexts as tags associated with the contacts because they are highly valuable information and according to Whittaker et. al: whether

Figure 2.6: Forget-me-not Implemented on The ParcTab Hardware

a contact is important to a user or not largely depends on the *history of their prior interactions* [57]. After investigating the design principles of context-aware systems and Forget-me-not, it is our conclusion that some important aspects should be apply to our context-aware Contact application as follows:

- *Internal vs. External Context*: External contexts being easier to capture, our plan is to focus on external contexts like date/time and location first. After that, we will try to extract internal context by accessing users' call logs, instant messages, emails and so on.
- *Data Capturing Methods*: The core of our system lays on the mobile phones with many available sensors like GPS, accelerometer, gyroscope. The operating systems of modern mobile phones allow applications to access the data gathered by those sensors directly so the capturing method we will mainly use is *Direct sensor access*.
- *Logical Architecture*: Following the architecture described by Ailisto et. al., our system will use four layers: Sensors, Raw data retrieval, Storage/management, and Application. The Preprocessing layer is omitted like the majority of context-aware systems. Regarding the Sensors layer, we will take a step-by-step approach by utilizing the physical sensors at the beginning then moving on to explore applicable virtual sensors later. Notably, many elements of these layers have already been implemented by the mobile phone's operating system. We can take advantage of that and focus on building the application.
- *Context Model*: We choose the key-value model because of its simplicity and high efficiency. The contexts we capture will be stored as numerous tags. The tags should be small and easy to search, retrieve, create, etc. Therefore, the key-value model is definitely one of the best choices for us.

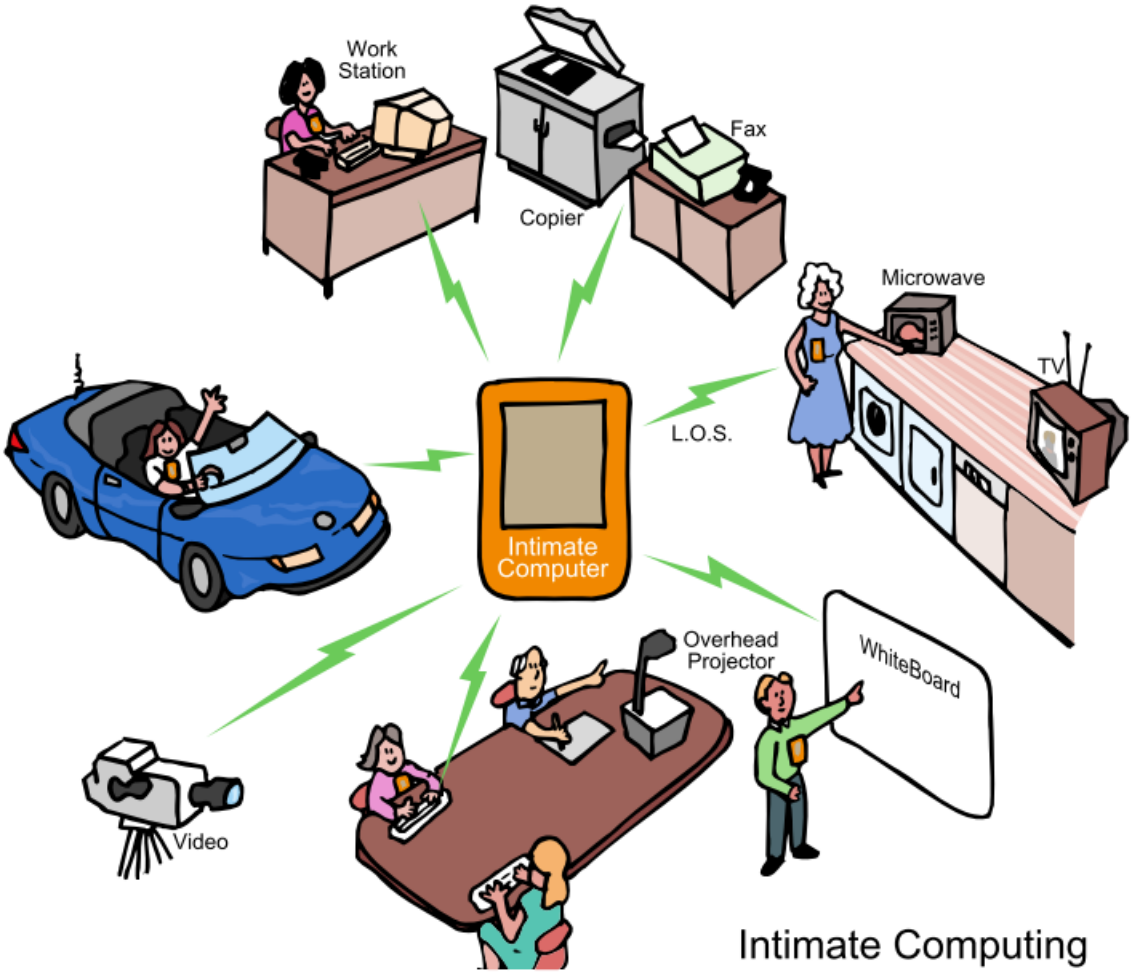

Figure 2.7: Intimate Computing Simple Model

- *Lessons from Forget-me-not Project*: What we are trying to do can be considered a small version of Intimate Computing. Our novel Contacts application will be a memory aid device which can help users find the name of a forgotten contact through the events having happened around him/her in the past or through the interaction history between the users and him/her. In the first phase, we will try to make our application collect some easy clues, activities which can help recall a contact like the date/time when the contact is created, the location when it happens. In future expansion, we will include more sophisticated clues like the event happening during the creation date, or the interactions between the users and the contacts.

2.2 Contact Relationships

There are various studies of the Contacts application from different aspects like improving the contact information, demoting unused/unimportant contacts. However, the relationships between contacts seem to have

been forgotten by researchers. In our opinion, contact relationship is an important characteristic which need to be investigated. In this thesis, we introduce a novel internal network of contacts where each relationship between two contacts is an edge of the network graph. All contacts in a mobile Contacts application and the relationships between them form a big network which we call a "Reversed Social Network". There are two reasons why we call it a reversed social network. First, it shares the same features with a normal social network. Second, the content in it comes from a reversed point of view. In other words, while in a normal social network like Facebook the social profiles are created by the actual attendees, in a reversed social network the profiles are created by the network owner - the user of the Contacts/Phonebook application. In this section, we will investigate the definition of a social network site then compare the similarities/differences between a reversed social network and a normal one.

2.2.1 Social Network Sites: Definition and History

Social network sites have been growing enormously, attracting hundreds of millions of users all over the world. In their work "Social Network Sites: Definition, History, and Scholarship" [29] Boyd and Ellison defined social network sites as web-based services which allow people to:

1. Build a public or semi-public profile in their bounded system.
2. Establish a list of other users who they have a connection with.
3. Explore and traverse their connection list and other users' list within the system.

Social network sites are unique not because they help people to meet new friends, but rather that they allow users to establish their own existing social networks online. Sometimes new connections between strangers are created but that is normally not the main goal. Connections in a social network site are often formed between participants who share some offline connections.

Although different social network sites have different features, the majority of them possesses a backbone of user profiles which illustrates a collected list of friends who are also users of that network. A profiles is a special page for an individual to "type oneself into being" [55]. Usually after joining a social network, a user is requested to fill out a form of questions. Then the answers in the form will be used to generate that person's profile page. Some typical questions ask about name, age, location, interests and a short description of the participant. People also have the option to upload a profile picture, and enhance their profiles by adding multimedia content. Profile visibility differs from site to site and according to user settings. By default, Facebook users can view other's profiles if they are directly connected, unless a user chooses to restrict his/her profile accessibility permission. On the contrary, LinkedIn decides what a user can access based on whether or not he/she has a paid account.

Many social network sites support a method for participants to leave messages on their friend's profile pages. This function often leads to another feature called leaving comments on these messages. Additionally, social networks usually have a private messaging system similar to email.

Besides user profiles, connections, comments, and private messages, social network sites differ vastly in functionalities and participants. Some sites have photos and videos sharing features while other ones have built-in blogging capability. A few social network sites are specified for mobile phones such as Dodgeball [41]. However, most web-based social networks also support mobile devices with some limitation like Facebook, LinkedIn, MySpace. Several social network sites are designed for users in particular geographic regions and languages. There are also some social networks which target particular religious, ethnic, political, or other personality-driven groups. Furthermore, social networks for dogs and cats even exist and their owners will be the one who manage their pet's profile.

2.2.2 Graphy vs. Social Network Sites

Similarities

According to the definition from Boyd and Ellision above, Graphy can be considered a special type of social network:

1. *Building a public or semi-public profile in their bounded system*: Every record in a Contacts application is a profile of an individual. A contact profile and a social network profile are strikingly similar. They both consists of basic information about the person and a profile picture. The difference here is that the contact profiles are mostly private and only visible to the owner of the application.

2. *Establishing a list of other users who they have a connection with*: In Graphy, we connect the contacts with each other based on their real life relationship.

3. *Exploring and traversing their connection list and other users' list within the system*: The owner/user of Graphy can view and traverse the entire network of connections.

From this comparison, we strongly believe that Graphy can stand out as another type of social network which may even evolve further like some other specialized services such as QQ, Cyworld, etc. QQ began as a Chinese instant messaging service which then developed into a major social network in China. Cyworld started as a Korean discussion forum, LunarStorm as a community website, Skyrock as a French blogging tool before including social network functionalities. Classmate.com was a school directory and started supporting friend list after social networks became widespread. All these services were founded and were based on functionalities not directly related to social network just like Graphy (based on Phonebook/Contacts application), then adapted to turn into full-feature social network sites.

Differences

Reversed Social Network The biggest difference between Graphy and a normal social network site like Facebook is the nature of the nodes and their connections in the networks. In a normal social network, a node is typically an user profile page which contains the information provided by that user. The connections

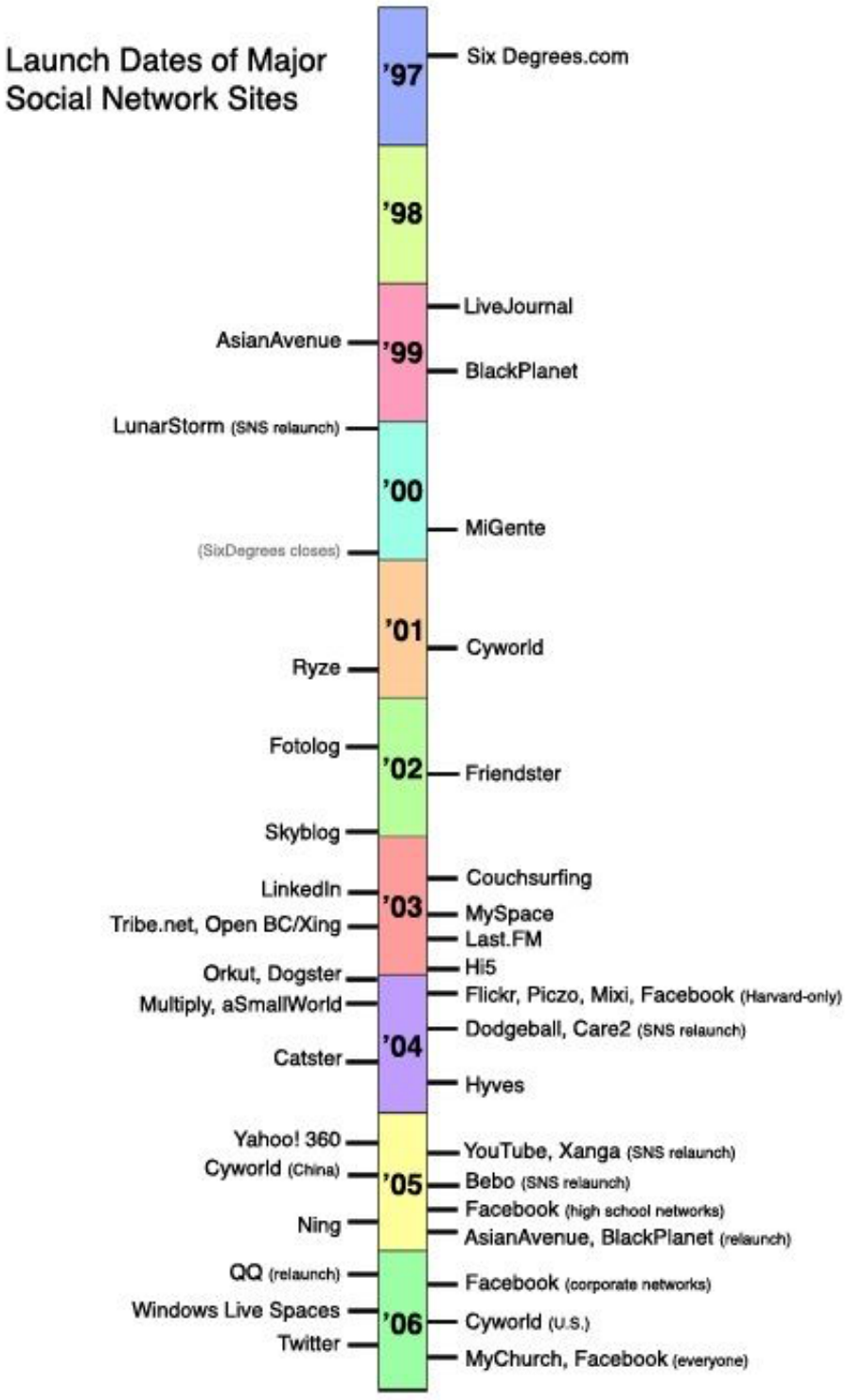

Figure 2.8: Timeline of the launch dates of various major social network sites and dates when community sites re-launched with social network features

between the nodes are established based on real life relationships which are also specified by the users. Therefore, the content of normal social networks comes from the "first-person perspective" or the owner of the nodes. On the contrary, in our Graphy system, the nodes are the contacts of the phone book/Contacts application. The information of the contacts as well as the relationships between them are determined by the owner of the phone book. In other words, the content of Graphy comes from the "third-person perspective" or the owner of the whole network.

Information Accuracy In online systems which allow people to freely assemble an online portrayal of self, there are several processes of impression management and self-presentation. Although most systems promote their users to create authentic representations of themselves, they usually do this to different degrees. In Friendster, an extremely popular social network site during the period between 2002 and 2007, many participants put incorrect information about themselves into their profiles. There were even profiles called "Fakester" which Boyd asserted that they never were real in her study [29]. Friendster was designed to only allow participants to view profiles of other users who were less than five degrees away. As a result, to extend their scope, people started adding acquaintances and strangers as friends so they can view more profiles. Some participants even tremendously collected friends. Not only serving the purpose of profiles viewing, the friends list was also an identity indicator of the profile owner. Consequently, people on social network sites had a high tendency to add interesting strangers into their circle of friends. Exploiting this trend, MySpace spammers created attractive fake profiles to collect targets for spamming. The problem of inaccurate friendships were emphasized even more in another study by Boyd [27] in which she pointed out that friends on social network sites are not the same as friends in real life and online social network friends present to offer people imaginary audiences to guide behavioral norms. To sum up, in a normal social network, user profiles and the relationships between them do not have a high accuracy.

In Graphy, the contact profiles and their relationships are established by the information given by Graphy's users. Therefore, the data is much more reliable than the ones in normal social network. The only one case where Graphy's data is not accurate is when the users enter wrong information which is less likely to happen. Comparing with normal social network sites, Graphy content is not abundant, however, it is much more accurate, useful to the users, and more compact with less irrelevant data.

Relationship After joining a social network, people are regularly asked to find other users who they have a relationship with. However, these relationships often do not provide much information about the real-life relationship between them. The labels for these relationships vary from site to site, the common terms are "Friends", "Contacts", and "Followers". Almost all social network sites use bi-directional relationships. There are a few sites allowing one-directional connections which are often labeled as "Followers" or "Fans", but some of them just use the term "Friends" like the majority. Notably, the term "Friends" is sometimes misleading since the relationships do not necessarily imply normal real-life friendship and there are many reasons behind people's social network connections. For example, LinkedIn only uses plain connections, Facebook uses

mostly "Friends" connections and a few special connections such as "Father", "Mother", "Brother", etc. Even though Facebook users can categorize their friends in different groups, it is still nearly impossible to describe a relationship like "A is a student of B", "B is the professor of A", etc. In a public environment like a social network, revealing these kinds of relationships may not be ideal since people more or less want to protect their privacy. In comparison, these types of relationships could be used frequently in a reversed social network where people can freely create these relationships between their contacts without any concerns of privacy.

Privacy The arrival of social network sites have caused an increasing concern for internet privacy. Since social network sites promote information sharing and collaboration many people are giving their personal information out on the internet. These social network sites usually keep track of all user interactions. This behavior leads to a number of issues such as "cyberstaking", social profiling, and government surveillance. In one of the first scholarly research of privacy and social network [39], the researchers studied 4000 Carnegie Mellon University Facebook profiles. They revealed potential privacy threats contained in the personal information published by the students on Facebook, for example attackers could build up users' social security numbers from public information on their profile page. On the contrary, there is no privacy concerns in a reversed social network. In contrast with the public nature of a social network, the nature of a reversed social network is private. The reversed social network is made to provide information to the network's owner so there is little sharing in it. In the future version of Graphy, we plan to implement a contact sharing feature which allow a Graphy user to share his/her own contacts with another user. This type of sharing is controlled and limited hence we believe no privacy threat will be exposed.

Bridging Online and Offline Social Networks An important aspect of social networks is that they support pre-existing social relations. Ellison et. al. [35] argue that Facebook helps its users maintain current offline relationships rather than meeting new friends. This is one of the main features which make social network sites stand out from earlier forms of public online community like forums and newsgroups. Many academic studies have been conducted on how online relations interact with offline ones. For example, Lampe et. al. [45] discovered that Facebook users spend much more time searching for someone they have an offline connection with than browsing for total strangers. Similarly, Boyd [28] suggested that MySpace and Facebook allowed U.S. youngsters to socialize with their friends even when they are unable to meet each other.

Being a reversed social network, Graphy also assists pre-existing offline connections. A relationship between two of Graphy's contacts represent their real-life connection. Moreover, a contact in Graphy should be someone the Graphy user needs to communicate with. Looking at this angle, Graphy connects the online and offline worlds even stronger than a normal social network does. The relationships in a normal social network are often weak ties [29] since a user's friend list often consists of many friends of friends, and people with some weak offline connections like being in the same organization. On the other hand, Graphy users generally just add a contact if they want to keep in touch with that person, this behavior indicates a stronger relationship.

2.3 Cloud-based Contacts Application

2.3.1 Overview of Cloud Computing

Cloud computing has been an evolving trend in the computing world recently. According to the U.S. National Institute of Standards and Technology [48], cloud computing is a model which facilitates convenient, ubiquitous, and on-demand network access to computing resources. The model consists of five fundamental characteristics, three service models, and four deployment models.

Fundamental Characteristics

- *On-demand self-service*: Users can independently control computing power as they need without directly contacting the service providers.
- *Broad network access*: The services are accessible over the network through miscellaneous platforms like personal computers, laptops, phones, and tablets.
- *Resource pooling*: The service providers' resources are pooled. According to user usage, physical and virtual resources are dynamically allocated.
- *Rapid elasticity*: The service providers can plan and supply computing capabilities in an elastic way to scale back and forth regarding consumer demands. From the user point of view, the capabilities seem to be unlimited and are ready at any time.
- *Measured service*: The resources can be controlled, monitored, and optimized automatically by the cloud system.

Service Models

- *Software as a Service (SaaS)*: The users consume the service by using the providers' applications. The applications can be provided through different interfaces like personal computer programs, mobile apps, or web browsers. All the underlying infrastructures like hardware, network, operating system and configuration are totally hidden from the end users.
- *Platform as a Service (PaaS)*: The users consume the service by implementing applications onto the cloud infrastructure using libraries and tools supplied by the service providers. Underlying infrastructures like hardware, network, and operating system are hidden from the users but they can control the configuration settings of the application environment.
- *Infrastructure as a Service (IaaS)*: The users consume the service by using the fundamental computing resource like processing power, storage to deploy and run any software. Underlying infrastructures

like hardware and network are hidden from the users but they can manage processing power, storage, operating systems and applications.

Deployment Models

- *Private Cloud*: The cloud infrastructure is provided to be used by a single organization. It could be owned and operated by that organization or a third party.
- *Community Cloud*: The cloud infrastructure is provided to be used by a specific community which have some common interests. It could be owned and operated by the community or a third party.
- *Public Cloud*: The cloud infrastructure is provided to be used by the public. It could be owned and operated by a company or an organization.
- *Hybrid Cloud*: The cloud infrastructure is a combination of two or more of the above cloud models.

2.3.2 Utilizing Cloud Computing in Contacts Applications

Traditionally, database of a Contacts application lies in separated phones. It makes the process of transferring contacts from phones to phones really difficult and prone to inconsistent data. Nowadays, with the support of cloud computing, there are many cloud-based database services for contacts like Outlook Contacts, Gmail Contacts. These services give users a lot of benefits as below:

- The ease to access contacts from anywhere
- Automatic synchronization contacts from phones to phones
- Backing up contacts
- Integrating phone book contacts with other services

Learning from that, we want our Graphy system to provide the same features. Unfortunately, all of the above services are closed source. Therefore, we decide to develop our own communication and synchronization solution which will be discussed carefully in the Implementation section 3. Using cloud computing models and the trending RESTful communication technology, our solution makes it easy for users to work offline through a local database while maintaining a central cloud database to backup and synchronize data to other devices when needed. Our cloud service model and deployment model will be software-as-a-service and community cloud. At the experiment stage, our application is used only by some targeted users but in the future, it is our hope that we can expand it into a public cloud service. Furthermore, the cloud server can be used to process long and heavy calculations for future features like relationships suggestion then push the results back to the clients.

2.4 Summary

This chapter investigates numerous aspects of the key areas of our problem. The summary of the related work that has been reviewed in this chapter is listed in table 2.2.

From the work having been reviewed, we conclude that modern Contacts applications still need a lot of improvements to help users manage and lookup their contacts better. Three key areas need improvements are: providing custom information to contacts, retaining historical context information, and establishing relationships between contacts. In the next chapter, we briefly describe a prototype of a new Contacts application which addresses these areas by introducing a number of enhancements.

Table 2.2: Summary of literature review

Design drivers for a modern Contacts application	• Contact management: identifying contacts to support long-term communication [57] • Designing for the evolution of mobile contacts application [43]
Users motivations and the use of tags in different existing systems, the implications for designing a tagging system in general, and how tags should be applied on mobile devices	• Usage patterns of collaborative tagging systems [38] • Why we tag: motivations for annotation in mobile and online media [25] • Users' needs for social tagging and sharing on mobile contacts [50]
Design principles of context-aware systems, and analysis of a specific context aware prototype	• A survey on context-aware systems [26] • Towards a better understanding of context and context-awareness [23] • Condor - an application framework for mobility-based context-aware applications [40] • User interactions with everyday applications as context for just-in-time information access [30] • Placing search in context: The concept revisited [36] • Structuring context aware applications: Five-layer model and example case [24] • Forget-me-not: Intimate computing in support of human memory [44]
Overview of social network sites and the comparison between a "reversed social network" and a normal one	• Social network sites: Definition, history, and scholarship [29] • Mobile social networks and social practice: A case study of dodgeball [41]
Fundamentals of cloud computing and the benefits of utilizing cloud computing in Contacts applications	• The NIST definition of cloud computing [48]

Chapter 3

Design and Implementation

Our proposed solution of Graphy is implemented into a working prototype running on iOS and Android devices which are backed up by a Unix-like server. In this section, some important features of Graphy as well as the implementation of those features are described.

3.1 System Architecture

As shown in Figure 3.1, the system contains 2 components: the mobile client and the backend server. The mobile client is an application which interacts directly with a user to capture his/her information. The backend server provides synchronization between multiple clients. Moreover, the backend server can perform additional processing like mining users' data, integrating with other services in the future. The communication between the client and the server is accomplished by the use of the RESTful architecture on top of HTTP/HTTPS protocols.

3.2 Mobile Client

3.2.1 Technologies

The goal of the mobile client is being a fully functional contacts management application. It should provide all nescessary features of a modern Contacts app as well as additional functionalities of Graphy. Moreover, it should run smoothly on popular smart phones. All things considered, we choose to use Xamarin - a state-of-the-art technology which makes it possible to do iOS and Android development in C#. The important thing is that Xamarin allows us to re-use and share a significant amount of code across the two device platforms. Compared with other cross-platform technologies like Sencha or Phonegap, Xamarin has many advantages namely:

- Native Performance: Xamarin code is compiled so it can leverage platform-specific hardware acceleration.
- Native User Interfaces: Xamarin utilizes the standard, native user interface elements so apps have different look-and-feel suitable for each platform.

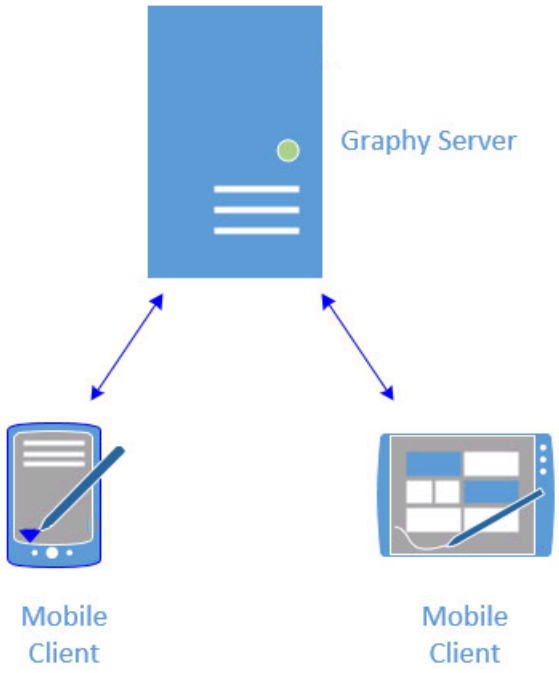

Figure 3.1: System Architecture

- Native API Access: Xamarin can access all the functionalities provided by the underlying platform and divice, including platform-specific capabilities.

In addition, the new Xamarin.Forms technology allows us to use the Model-view-viewmodel architectural pattern and share nearly 100% code on both platforms. The Model-view-viewmodel pattern is our preferred way of separating the graphical user interface and the business logic. Its two-way data bindings automatically synchronize data between controls and models, which is very suitable for our application. The architectures of Xamarin and the Model-view-viewmodel pattern can be found in Figure 3.2 and Figure 3.3.

3.2.2 Features

The mobile client has all normal features from a modern contacts management application such as adding/editting/removing contacts, searching for contacts, marking contacts as favorite. Additionally, to make the transition from users' old Contacts application to Graphy effortless, we import all contacts in their phone to Graphy the first time they run our app.

More importantly, the feature that distinguishes Graphy from other Contacts applications lies on the contact profiles. As we can see from Figure 3.5, a Graphy's contact profile consists of usual information

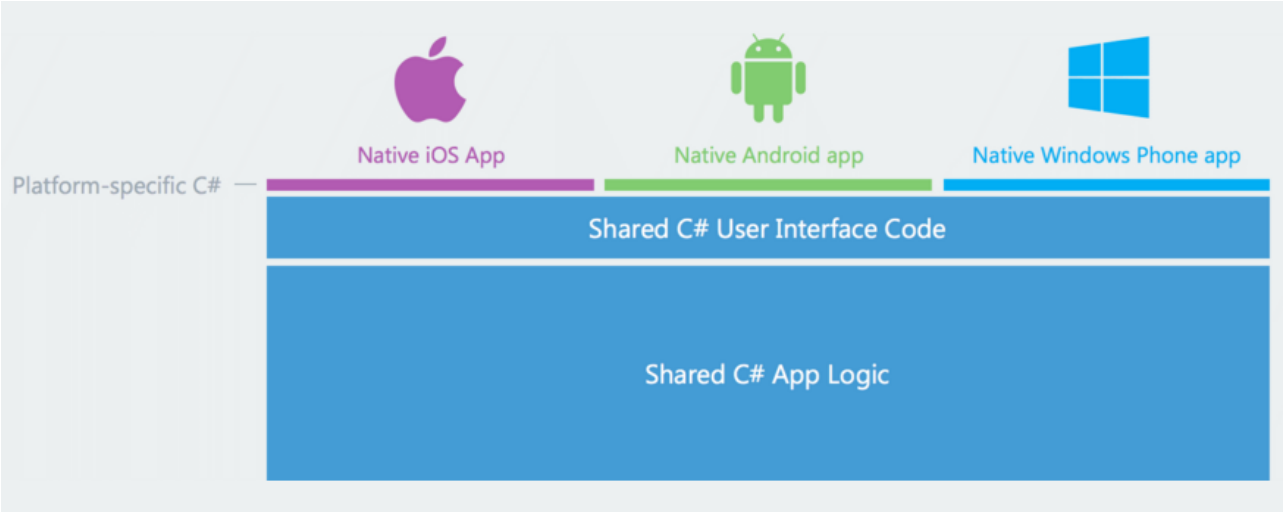

Figure 3.2: Xamarin Architecture

Reprinted from Xamarin.com, 2016, Retrieved from *www.xamarin.com/platform*

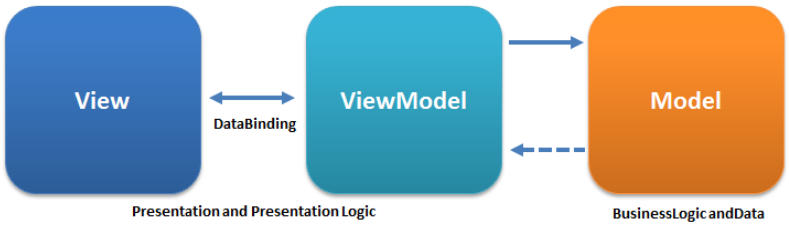

Figure 3.3: Model-view-viewmodel Pattern

Reprinted from Wikipedia.org, 2016, Retrieved from *en.wikipedia.org/wiki/Model-view-viewmodel*

plus the tags and the relationships. In this example, "Chairman of Microsoft" is a custom tag added by the user. The custom tags are implemented by 255-character text fields that cover most common use cases, and users can create as many tags as they want to. Furthermore, when a new contact is created, the date and the current location are automatically added as tags in the new contact's profile. These are two historical contexts which may help the user recall information about the new contact when he/she looks it up again in the future. In later versions, we will add more meaningful historical context tags to assist users better. Regarding relationships between contacts, in Graphy's contact profile, a relationship is demonstrated by 3 fields: the name of the relationship, the related person, and an optional detail about the relationship. In Figure 3.5, Bill Henry Gates is the advisor of Satya Nandela. This relationship is shown by the "=>" symbol. Conversely, Jennifer Gates is the daughter of Bill Henry Gates so the symbol is reversed to "<=". These symbols are not the most attractive way to represent the relationships and we will design a better user interface in later prototypes. It is worth mentioning that the relationships are bi-directional. When the users tap on the relationships field, Graphy will bring them to the other contact. For example, tapping on Satya Nandela will display the full contact profile of Satya in which there is a relationship "Advisor" from Bill Henry

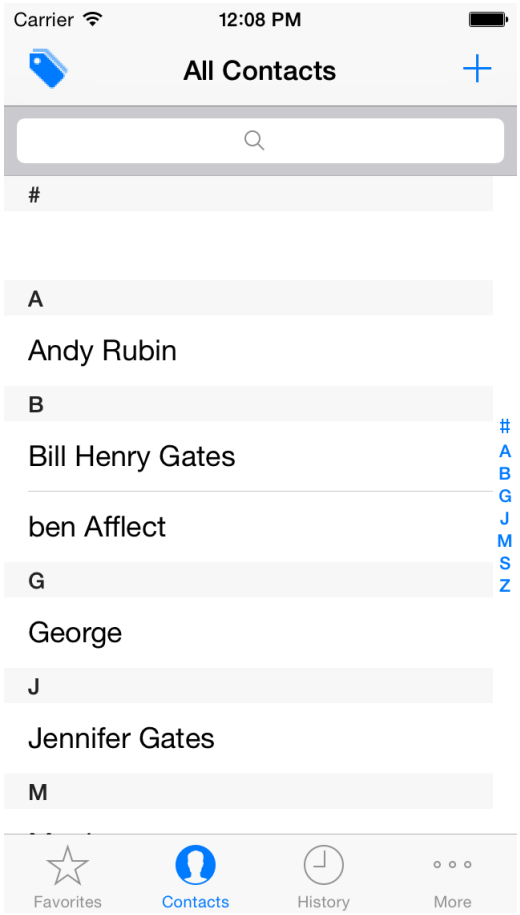

Figure 3.4: Main Screen

Gates. Users can tap infinitely to navigate and explore their relationship networks. Besides, all fields of a relationship are customizable.

In addition to the new elements in the contact profiles, we also introduce new ingredients to the search function of the mobile client. Graphy is not only capable of searching for contacts via traditional information like names, organizations but it also supports searching by tags and relationships. Users can enter keywords to Graphy's search boxes then the application will look for that pieces of information in traditional fields, tags, and relationships of all the contacts. Likewise, users can utilize the "relationships tapping" feature to jump between contacts and find the people they are looking for.

The mobile client is fully functional in the offline mode thanks to the implementation of a local SQLite database. When the client connects to the backend server it can synchronize its local database with the server. This design provides the users with the ability to use multiple devices concurrently.

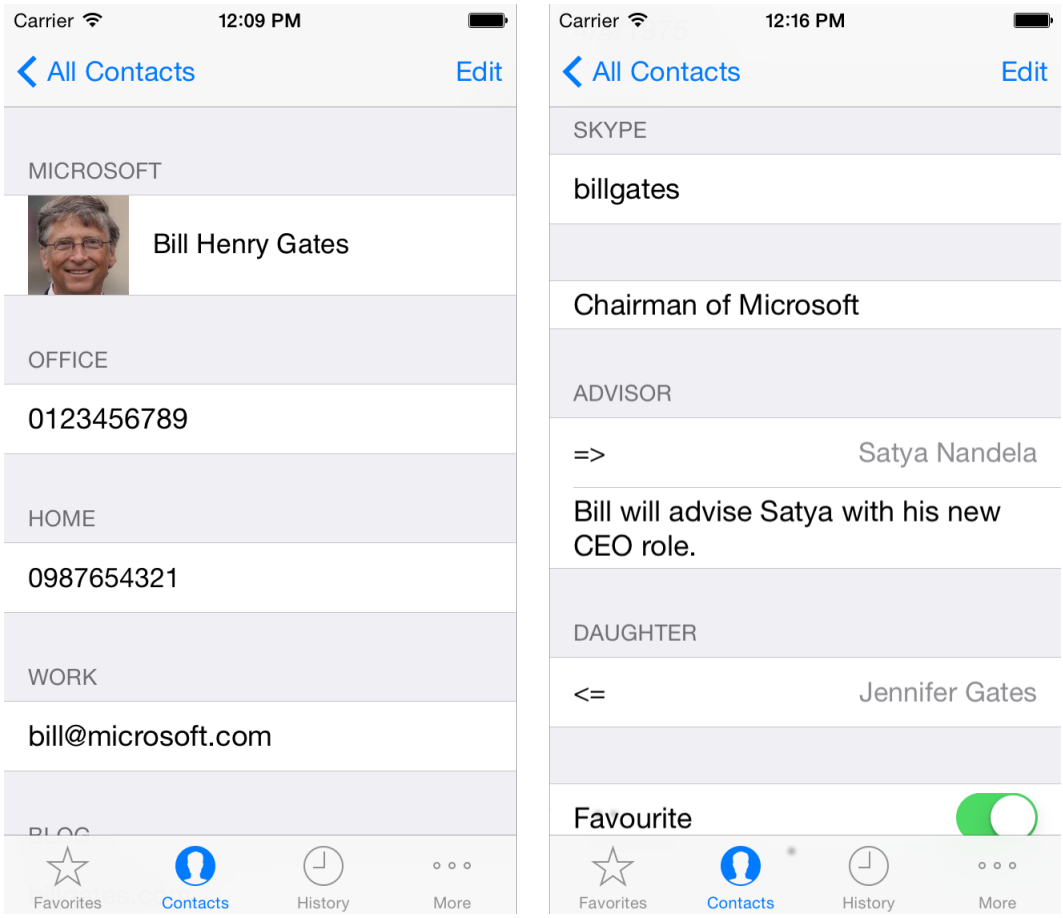

Figure 3.5: Contact Profile

3.3 Backend Server

The goals of the backend server are providing a synchronization service for users on multiple devices, and being a foundation infrastructure for future calculation tasks. One important requirement for the backend server is that it should communicate with the mobile clients through a common and simple protocol. Given these points, we choose HTTP as the protocol, Python as the programing language, Django as the web framework. The Hypertext Transfer Protocol or HTTP is an application protocol for distributed, collaborative, hypermedia information systems [46]. It is widely used and is the foundation of the World Wide Web's data communication. HTTP operates as a request-response protocol in the client-server computing model. Coupled with HTTP, the Representational state transfer or REST is a state-of-the-art architectural style for designing networked applications. REST can use HTTP for all of its operations. It is a lightweight alternative to Remote Procedure Call and other webservices. In spite of its simplicity, REST is very robust and there is basically nothing we can do in a webservice that cannot be done with REST. Therefore, using REST with the HTTP protocol becomes an excellent choice for our system's communication. Table 3.1 illustrates some example requests of the RESTful API which are exposed by the server. On the other

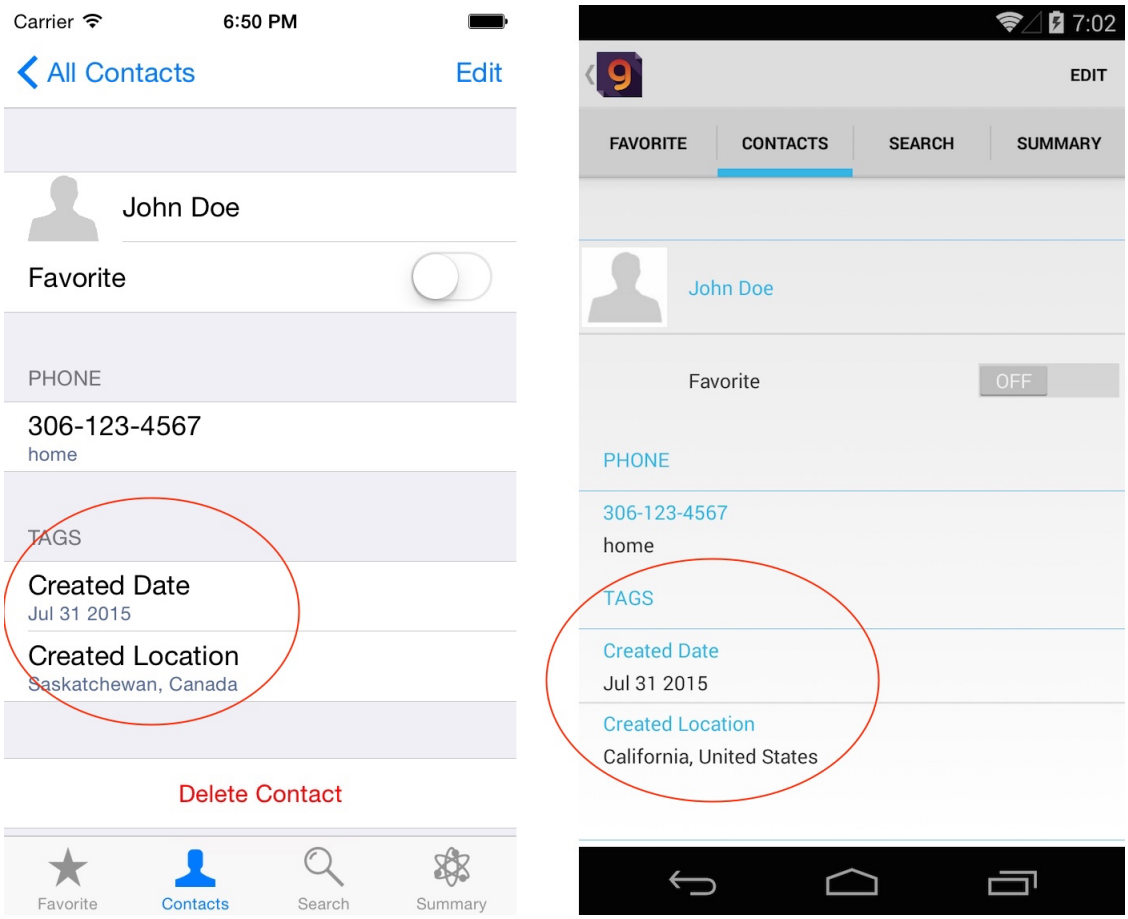

Figure 3.6: Historical Context Tags

hand, Python is our language of choice since we are familiar with it and it is a powerful, general-purpose programing language. Moreover, the Django web framework, which is written in Python, is an extremely popular open-source framework. It encourages rapid development and clean, pragmatic design while also being fast, secure, and scalable. Django helps us take care of many aspects in building a RESTful API so we can focus on the logic of the system. As regards the server, we decide to use a Unix-like local system to run the Gunicorn HTTP server which in turn hosts the Django framework. We prefer this solution instead of using a platform-as-a-service cloud system like Google App Engine or Microsoft Azure because we will have full control of the server and we can avoid some unnecessary cost during the development phase. Later we can rent a server instance from an infrastructure-as-a-service provider like Amazon EC2 to host our system on cloud.

3.4 Database Design and Synchronization

The databases play an important role in the system of Graphy. Since Graphy operates in both online and offline modes, the local database in a mobile client must contain all user's data. Furthermore, Graphy supports

Table 3.1: RESTful API

Verb	Request	Description
GET	/contacts/	Retrieve all contacts
POST	/contacts/	Create a new contact
GET	/contacts/{id}	Retrieve a contact with a specific Id
PUT	/contacts/{id}	Update a contact
DELETE	/contacts/{id}	Delete a contact

multiple devices concurrently so the databases on clients should synchronize flawlessly and efficiently.

3.4.1 Database Schema Design to Support Tagging

The problem of tagging items has been around in industrial products for a while. It was popularized by websites associated with Web 2.0 and is an important feature of many Web 2.0 services. In 2003, Delicious - a social bookmarking website provided a service for its users to add tags to their bookmarks as a way to help find them later [3]. Flickr [5] also let users tag their photos with custom information thus constructing flexible and easy metadata that enriched photos' information and made them highly searchable at the same time. Influenced by the success of Flickr and Delicious' popularized concept, many websites like Youtube [22], Gmail [7], StackOverFlow [14] also implemented tagging. Furthermore, the widespread microblogging service Twitter [18] introduced the hashtags which let its users tag their own "tweet" easily by prefixing the tagged words with a hash symbol. The idea became widely popular and as a result Twitter became a powerful search engine for searching trends or news all over the world. Not only online web services used tags, the operating system OS X version 10.9 also provided colorful tagging labels [10] that help users to tag their own files.

In order to provide a powerful tagging system (relationship is considered a special type of tag), we need an appropriate database schema. From the database point of view, the tagging problem can be defined as follow: we want to have a database schema where we can tag an item with as many tags as we want. Later on, we want to run queries to constrain the items to a union or intersection of tags. We also want to exclude/minus some tags from the search result [15]. There are three popular different solutions with which we will examine as follows:

MySQLicious Solution

MySQLicious is a library which provides automated mirroring/backups of Delicious bookmarks into a MySQL database [9]. The schema of MySQLicious has only one table as shown in Table 3.2

In this schema, the table is denormalized. The retrieving queries that can be applied on this schema are

Table 3.2: MySQLicious Schema

id	url	tags
1	`http://www.ics.uci.edu/~fielding/pubs`	api rest research
2	`http://stackoverflow.com/questions/983030`	coding .net
3	`http://www.ford.com/cars/mustang/`	cars wishlist

as follows:

Intersection

Query for "webservices+rest+coding":

SELECT *
FROM 'MyTable'
WHERE tags LIKE "%webservice%"
AND tags LIKE "%rest%"
AND tags LIKE "%coding%"

Union

Query for "webservices|rest|coding":

SELECT *
FROM 'MyTable'
WHERE tags LIKE "%webservice%"
OR tags LIKE "%rest%"
OR tags LIKE "%coding%"

Minus

Query for "webservices+rest-coding":

SELECT *
FROM 'MyTable'
WHERE tags LIKE "%webservice%"
AND tags LIKE "%rest%"
AND tags NOT LIKE "%coding%"

The advantages of this solution are:

- Just one table.
- The retrieving queries are very straight forward.
- Can achieve results via full-text search which can be a little faster.

The disadvantages are:

- We have a limit on the number of tags per item according to the size of the tags field. Normally we use a 256-byte field in our database (VARCHAR). Otherwise, if we use a text field or the like, the queries will become slow.
- Doing processing on tags is hard and expensive. For example, it is difficult to provide suggestions on a new tag. Therefore, synonymous tag names like "coding" and "programing" have a high chance of happening.

Scuttle Solution

Similar to Delicious, Scuttle [11] is a web-based social bookmarking system. It allows multiple users to store, share and tag their favorite links online. Scuttle organizes its data in two tables as shown in Figure 3.7. The table scCategories is the tag-table and has got a foreign key to the bookmark-table.

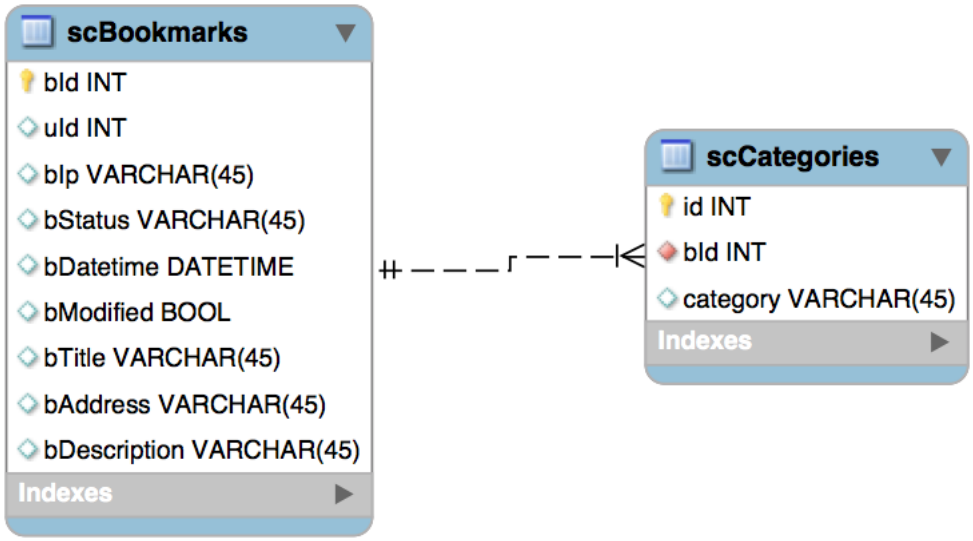

Figure 3.7: Scuttle Schema

The retrieving queries we can apply on this schema are as follows:

Intersection

Query for "webservices+rest+coding":

SELECT b.*

FROM ‘scBookmark’ b, ‘scCategories’ c

WHERE b.bId = c.bId

AND (c.category IN (‘webservices’, ‘rest’, ‘coding’))

GROUP BY b.bId

HAVING COUNT(b.bId)=3

Firstly, all combinations of bookmarks and tags are searched, where the tag is “webservices”, “rest” or “coding” (c.category IN (‘webservices’, ‘rest’, ‘coding’)), then just the bookmarks that have all three tags are selected (HAVING COUNT(b.bId)=3).

Union

Query for “webservices/rest/coding”:

SELECT b.*

FROM ‘scBookmark’ b, ‘scCategories’ c

WHERE b.bId = c.bId

AND (c.category IN (‘webservices’, ‘rest’, ‘coding’))

GROUP BY b.bId

Minus

Query for “webservices+rest-coding”:

SELECT b.*

FROM ‘scBookmark’ b, ‘scCategories’ c

WHERE b.bId = c.bId

AND (c.category IN (‘webservices’, ‘rest’))

AND b.bId NOT

IN (SELECT b.bId FROM scBookmarks b, scCategories c WHERE b.bId = c.bId AND c.category = ’coding’)

GROUP BY b.bId

HAVING COUNT(b.bId) =2

The advantages of this solution are:

- The schema is more normalized than the MySQLicious solution.
- Listing and doing processing on all tags in the system is rather easy.

The disadvantages are:

- The retrieving queries have JOIN queries which might be more expensive.
- There are many duplicated tags in the system since one tag record in the tag-table only associates with one item in the bookmark-table.

Toxi Solution

Toxi [17] an open-source software contributor came up with a three-table schema as shown in Figure 3.8. The bookmarks and the tags are n-to-m related by making use of a tagmap table. As a result, we can use one tag for many bookmarks and vice versa. This structure is also used by the popular blogging platform WordPress [21].

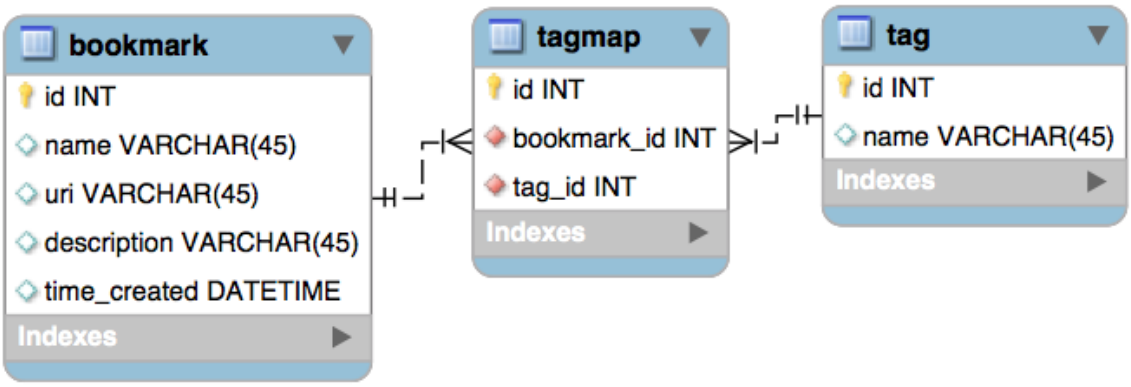

Figure 3.8: Toxi Schema

The retrieving queries we can apply on this schema are as follows:

Intersection

Query for "webservices+rest+coding":

```
SELECT b.*
FROM tagmap bt, bookmark b, tag t
WHERE bt.tag_id = t.id
AND (t.name IN ('webservices', 'rest', 'coding'))
AND b.id = bt.bookmark_id
GROUP BY b.id
HAVING COUNT(b.id)=3
```

Union

Query for "webservices/rest/coding":

```
SELECT b.*
FROM tagmap bt, bookmark b, tag t
WHERE bt.tag_id = t.id
AND (t.name IN ('webservices', 'rest', 'coding'))
AND b.id = bt.bookmark_id
GROUP BY b.id
```

Minus

Query for "webservices+rest-coding":

```
SELECT b.*
FROM bookmark b, tagmap bt, tag t
WHERE b.id = bt.bookmark_id
AND (t.name IN ('webservices', 'rest'))
AND b.id NOT IN (SELECT b.id FROM bookmark b, tagmap bt, tag t
WHERE b.id = bt.bookmark_id AND bt.tag_id = t.id AND t.name = 'coding'
GROUP BY b.id
HAVING COUNT(b.id) =2
```

The advantages of this solution are:

- The schema is in the Third Normal Form. It is the most normalized solution of all three.
- We can add extra information to each tag such as description, tag hierarchy.
- Doing processing on tags is easy.
- Flexible, scaling well.

The disadvantages are:

- The retrieving queries are quite expensive and can become very complicated. However, we can break down a complicated query and utilize the cache.
- Altering or deleting bookmarks can lead to orphan tag unless we use cascading.

Schema Performance Analysis

Test Setups Eachhome [16] did performance tests on all the solutions mentioned above: MySQLicious, MySQLicious with full-text search, Scuttle and Toxi. The test setups are as follows:

For each schema, there are 4 databases of 1000, 10000, 100000 and 1 million bookmarks. The tags associated with the bookmarks were random English words. Each bookmark got 1 to 10 tags attached to it. Every schema had exactly the same data. Then each schema was queried with an alternately 1-3 tag query. For example, the first query was "webservice", the second was "webservice+rest", the third was "webservice+rest+coding", the fourth again was just one tag like the first one and so forth. Each query was done with LIMIT 50. All the queries worked and the outcomes were the same on all schemas.

The databased was MySQL 4.0.21 with the configuration described in Table 3.3.

Detail of the computation system is described in Table 3.4.

Table 3.3: MySQL Configuration

key_buffer = 300M
query_cache_size = 30M
query_cache_limit = 30M
table_cache = 64
ft_min_word_len = 2
ft_stopword_file = ''

Table 3.4: System Configuration

CPU	3GHz Dual Xeon
Cache	1MB
Harddisk	SCSI Ultra 320 Atlas 10K, no RAID
RAM	3GB

Test Results The first test was done with a tag set of 250 tags. As we can see from Figure 3.9, when the tag number of items was smaller than 10 thousands, all schemas performed very well. The differences in performance only appeared when the dataset was 1 million or more. When the number of bookmarks was 1 million, MySQLicious was very slow because of the filter instruction *WHERE tag LIKE "%tagname%"*. That was, MySQLicious had to go through the whole dataset to test each item against the queries. Moreover, applying full-text search did not make this solution any faster. On the contrary, Toxi solution was really fast. It was about twice as fast as the Scuttle solution and four times faster than MySQLicious.

The second test was done with a 999 tag set. Looking at what happened in Figure 3.10, MySQLicious with full-text search was the performance leader. It indicated that if we have a system with diverse tag distribution, MySQLicious with full-text search will be the best solution. According to Eachhome research [16], if a system has an average tag distribution of 1% (a tag shows up in 1/100 of all items on an average), Toxi is the best solution. On the other hand, if the system is more uniformly ditributed, MySQLicious performs better.

Regarding Union queries, MySQLicious was the fastest as we can see in Figure 3.11. Taking advantage of the *LIKE* queries, MySQLicious sought through all the items then harvested the items with one of the given tags. The magic here lay in the *LIMIT* instruction. MySQLicious traversed the items, checked the constraints then stopped after finding enough items as specified in the *LIMIT* instruction. Whereas in other solutions, the database had to join the tags with the items first then searched through the join result, thus increasing a significant amount of time.

Compared with intersection and union queries, insertion queries are remarkably faster. In Figure 3.12, the scale of the vertical-axis changed to 0 - 2 millisecond. MySQLicious handled insertion queries very well, its variation of full-text had to create the full-text index and therefore was a little slower. The Scuttle and

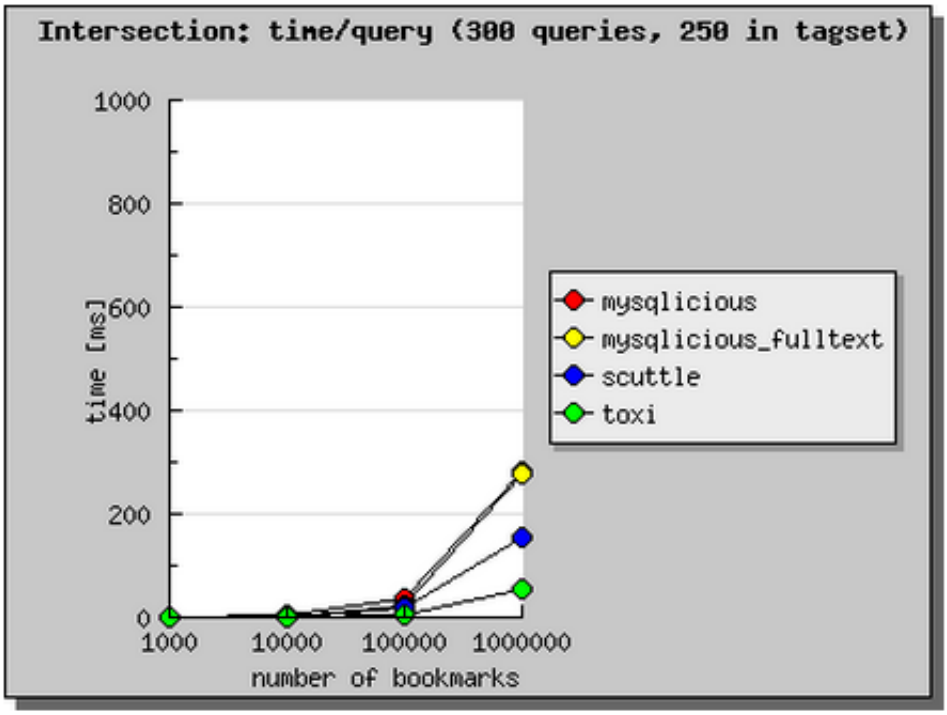

Figure 3.9: Intersection Queries with 250 Tag Set

Toxi solutions are slower because of the creation of the second and the third tables in their schema. However, insertion queries in all schemas were still about 100 times faster than intersection and union ones. Therefore, we should base our schema decisions on intersection and union queries.

Improving Performance There are four popular ways to improve the performance. First, we can use *Caching*, we can cache the results of the recent queries for some hours or cache the items per tag. Second, we can use both MySQLicious full-text and Toxi solutions at the same time then use the appropriate schema based on the characteristic of the queries. For example, simple union queries will be performed on MySQLicious while intersection queries with common tags on Toxi. Third, we can slice the data to user/tag/item and prebuild some results. Forth, we can use a NoSQL database. However, using a NoSQL database will lead to other requirements which need to be taken care of.

Conclusion

The nature of a Contact application requires consistency on a small dataset (less than 1 million records). Therefore, we choose the Toxi [17] solution which has been discussed above. The local database on a mobile client uses SQLite. It processes all user's queries locally then automatically synchronizes with server database when there is an Internet connection. The server uses a PostgreSQL database with a slight change in the schema to store data from all users. Parts of the database schema are shown in Figure 3.13 and Figure 3.14.

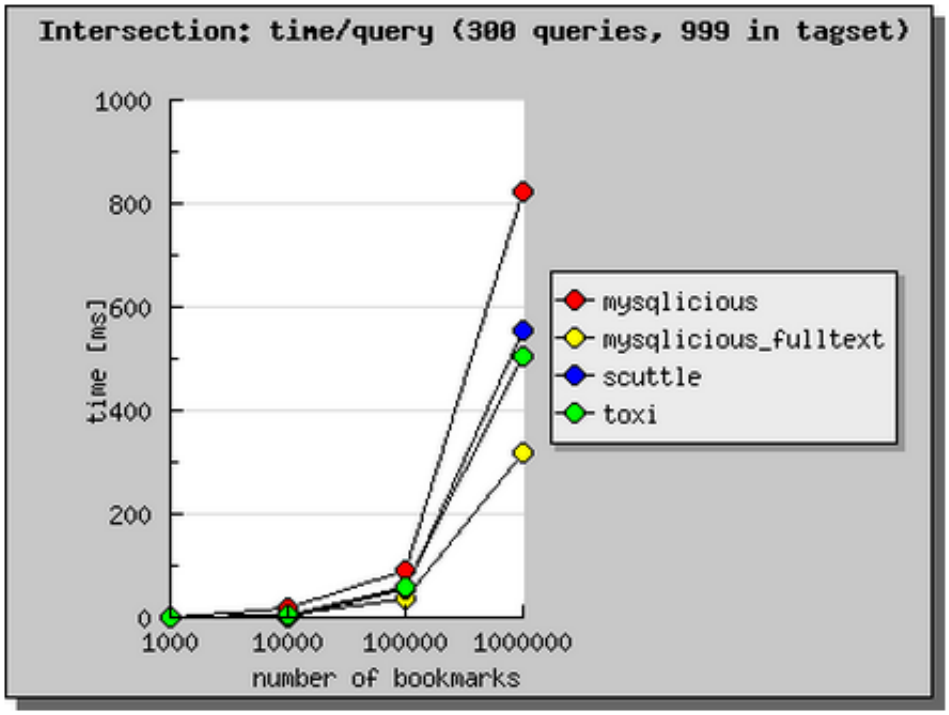

Figure 3.10: Intersection Queries with 999 Tag Set

3.4.2 Synchronization Technique

In order to provide an efficient synchronization solution, each Graphy's mobile client maintains a Sync Queue in its local database. When the user creates, updates or deletes a record, the operation will be applied immediately to the local database and at the same time that operation will also be appended to the Sync Queue. When the mobile client has Internet connection, Graphy will periodically communicate with the backend server to sync the current operations in the Sync Queue to the database on server. Since our work is not focusing on designing a synchronization algorithm, we make an assumption that the mobile clients and the backend server are always having the same clock. Obviously, the clock synchronization process can be done by applying a well-known method such as Network Time Protocol, TLSDate, or using versions.

The synchronization process is briefly described as follows:

1. Client sends GET requests to all endpoints of the API. The server returns all records of that user. After that, the client examines the server's records, if they have the last_modified field greater than the client's ones, the client will update its database and Sync Queue according to the information from the server.

2. Client consecutively performs all operations in its Sync Queue:

   (a) If the operation is a creation:

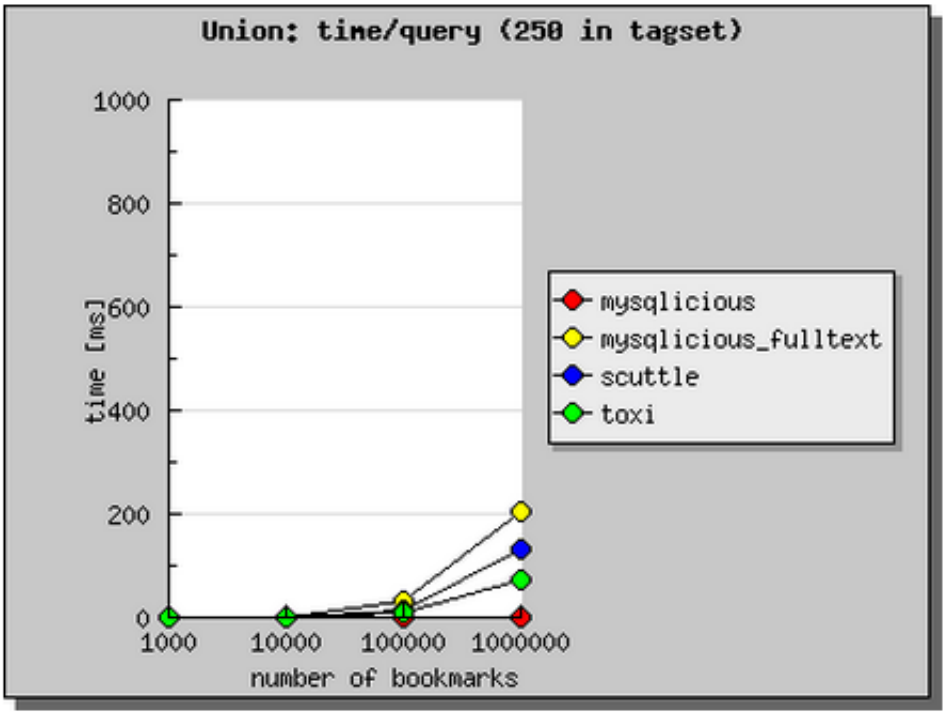

Figure 3.11: Union Queries with 250 Tag Set

i. Client sends a POST request to the server with detail of the new *client record* in the request's body.

ii. Server creates a new *server record* according to the request's body.

iii. Server sends back to client an HTTP 201 CREATED response which contains the *server record*.

(b) If the operation is an update:

i. Client sends a PUT request to server with details of the updated *client record* in the request's body.

ii. Server examines the *client record*. The server will return one of the three HTTP responses: 204 NO CONTENT, 409 CONFLICT, or 410 GONE after comparing the *client record* and the *server record*.

iii. Client receives the server response and bases on it to make decision on keeping, updating or deleting the *client record*.

(c) If the operation is a deletion:

i. Client sends a DELETE request to server with the ID of the *client record* and an IF-UNMODIFIED-SINCE header which is set to be equal to the record's last_modified field.

ii. Server examines the *client record* and compare it with the *server record*. After that, it will decide to mark the *server record* deleted or not, then return one of the three HTTP responses:

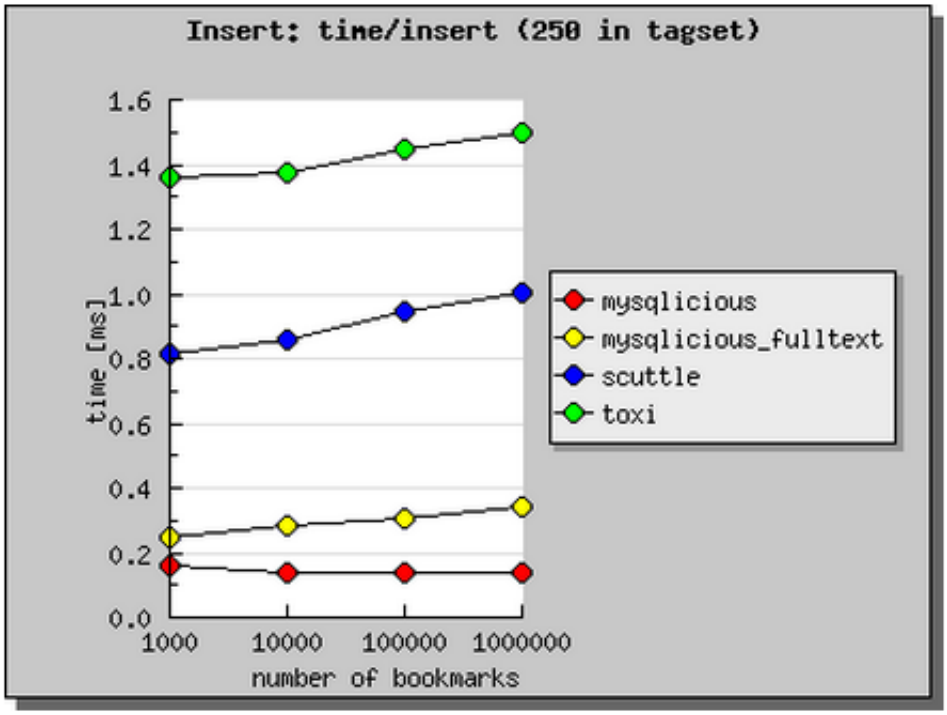

Figure 3.12: Insertion Queries with 250 Tag Set

204 NO CONTENT, 409 CONFLICT, or 410 GONE.

iii. Client receives the server response and bases on it to make decision on keeping or deleting the *client record*.

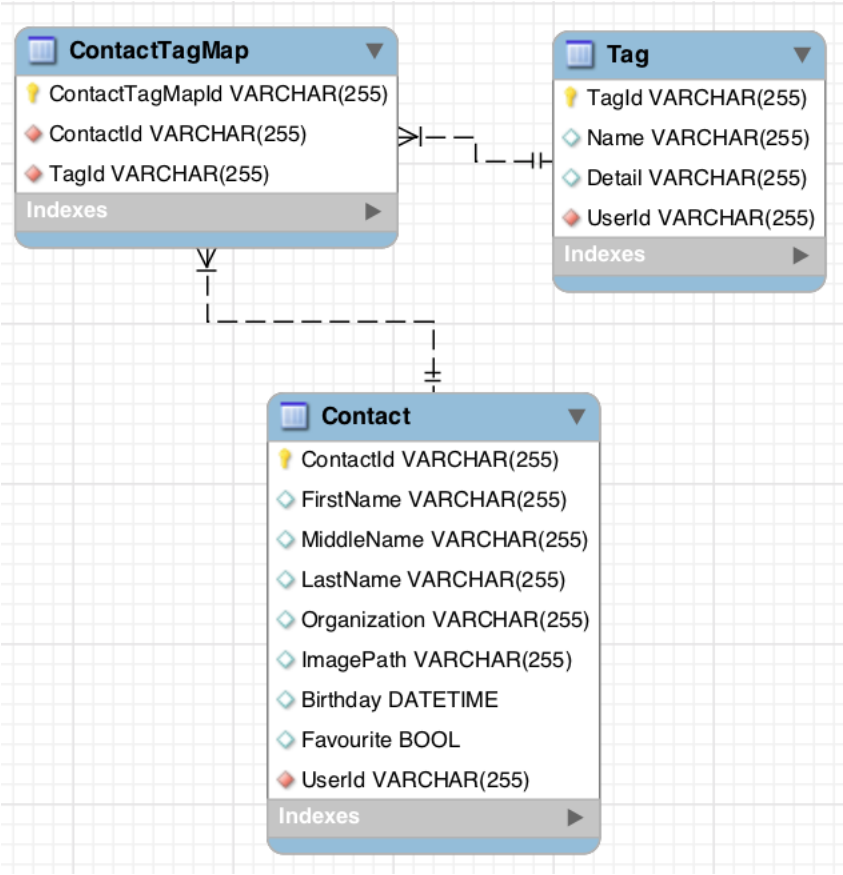

Figure 3.13: Tag Schema

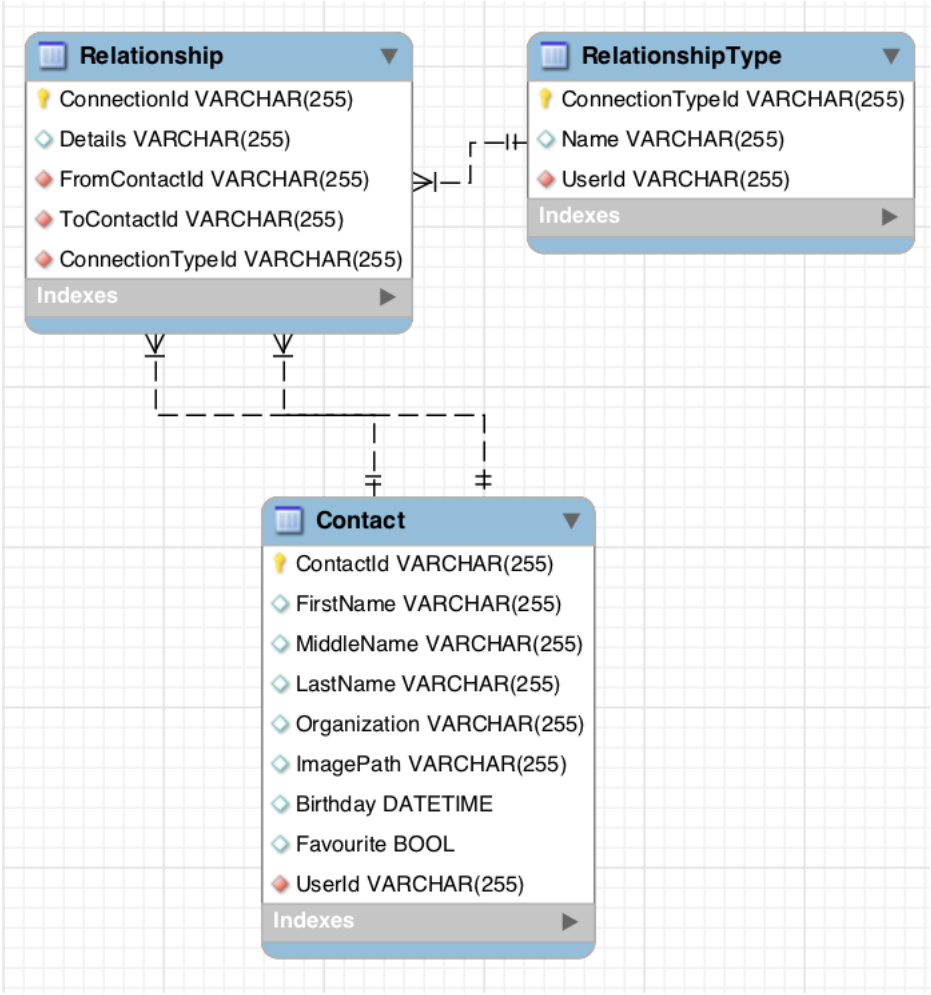

Figure 3.14: Relationship Schema

Chapter 4

Experiment and Evaluation - User Study

In this chapter, the Graphy prototype is evaluated in a user study. The objective of the study is to evaluate the user acceptance of Graphy to see whether it achieves its three goals: allowing users to find contacts by their miscellaneous information, efficiently retain knowledge of contacts, and enabling users to establish relationships between contacts then traverse the relationships with ease. The chapter contains three sections describing the hypotheses of the study, the detailed experimental setup, and the results.

4.1 Hypotheses

The main goal of this user study is to evaluate Graphy's functionalities in real use over a period of time. The evaluation mainly aims at testing the following hypotheses:

1. Graphy allows users to find contacts by their miscellaneous information. Users often use this functionality.

2. Graphy allows users to efficiently retain knowledge of contacts. Users utilize this feature to maintain a considerable amount of information, especially miscellaneous information.

3. Graphy allows users to establish relationships between contacts then traverse the relationship network with ease. Users create a lot of relationships and actively traverse the relationships during their usage.

4.2 Experimental Setup

The method of this user study is to recruit people to download the Graphy application to their smart phones after signing a consent form to participate in the study. The participants then use it in an uncontrolled environment for two weeks. The Graphy application records users' activities like creating, editing, and searching for contacts into a list of numbers on a summary screen which is part of the user interface. After using it for two weeks, the users go to the summary screen and copy all the numbers on it to an online form. The summary screen and the online form are illustrated in figures 4.1) and 4.2).

To recruit participants, we sent email invitations to our contacts list and also advertised the study on Facebook. As a result, twenty one people installed the application and finished the online form. The form

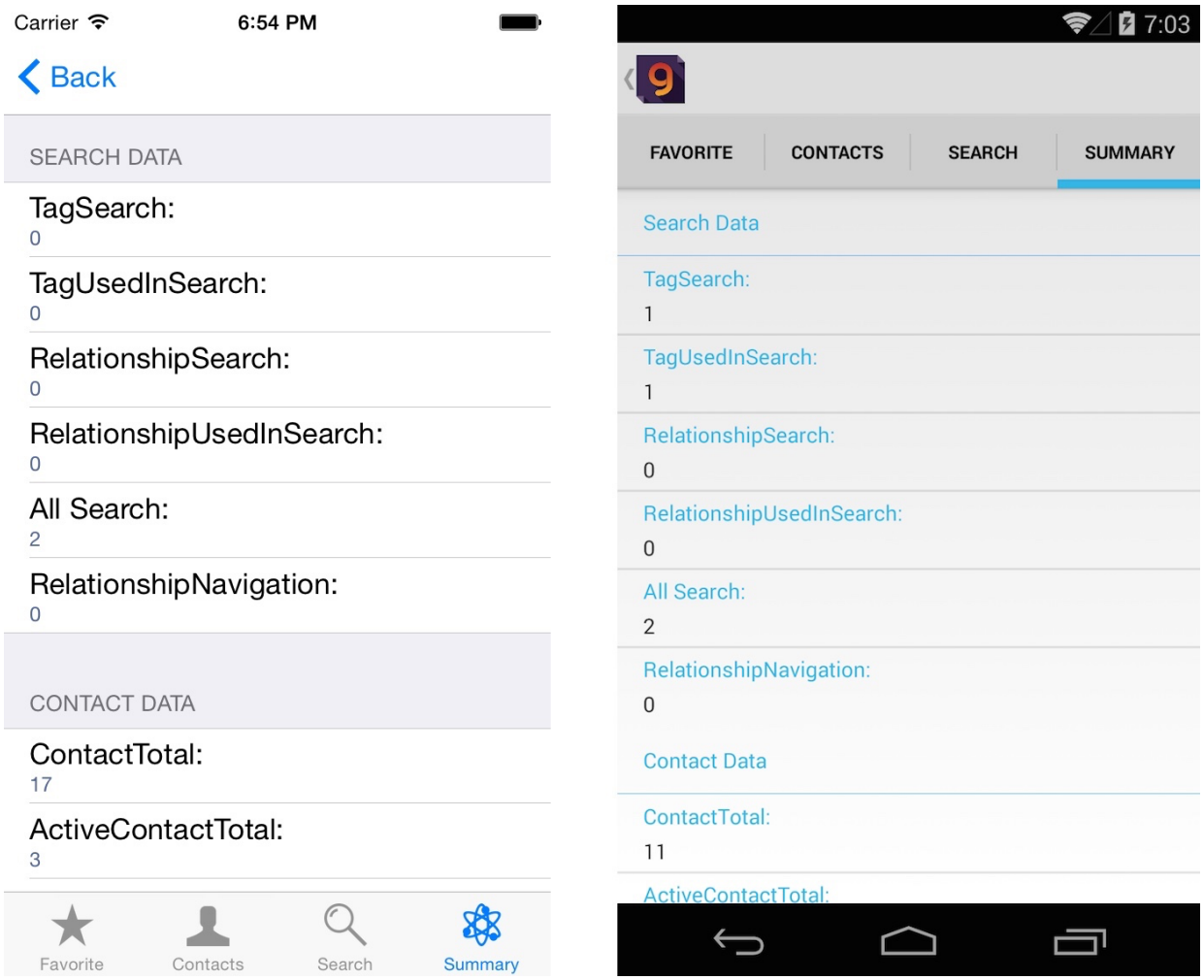

Figure 4.1: Graphy's Summary Page

is divided into two categories: one for tags (which represent pieces of miscellaneous information), one for relationships. Each category has two sections: the contact data and the search pattern. The contact data section accumulates information in the users' contacts list. For example, it records the total number of contacts, the total number of tags and relationships created during the study. This section allows us to examine the way participants store their information using Graphy, how they utilize the newly introduced elements in our application. The search pattern section collects data when a participant uses the search function or traverse the relationships network. For example, it tracks the frequency a user searches for contacts using tags, the number of tags used in each search, and how many times the user navigates from one contact to another through the relationship network. This section helps us to understand how people consume the data they store in Graphy: whether they use the newly introduced elements to search or they still use traditional information like first name and last name, whether they use the relationships network to look for a contact, etc.

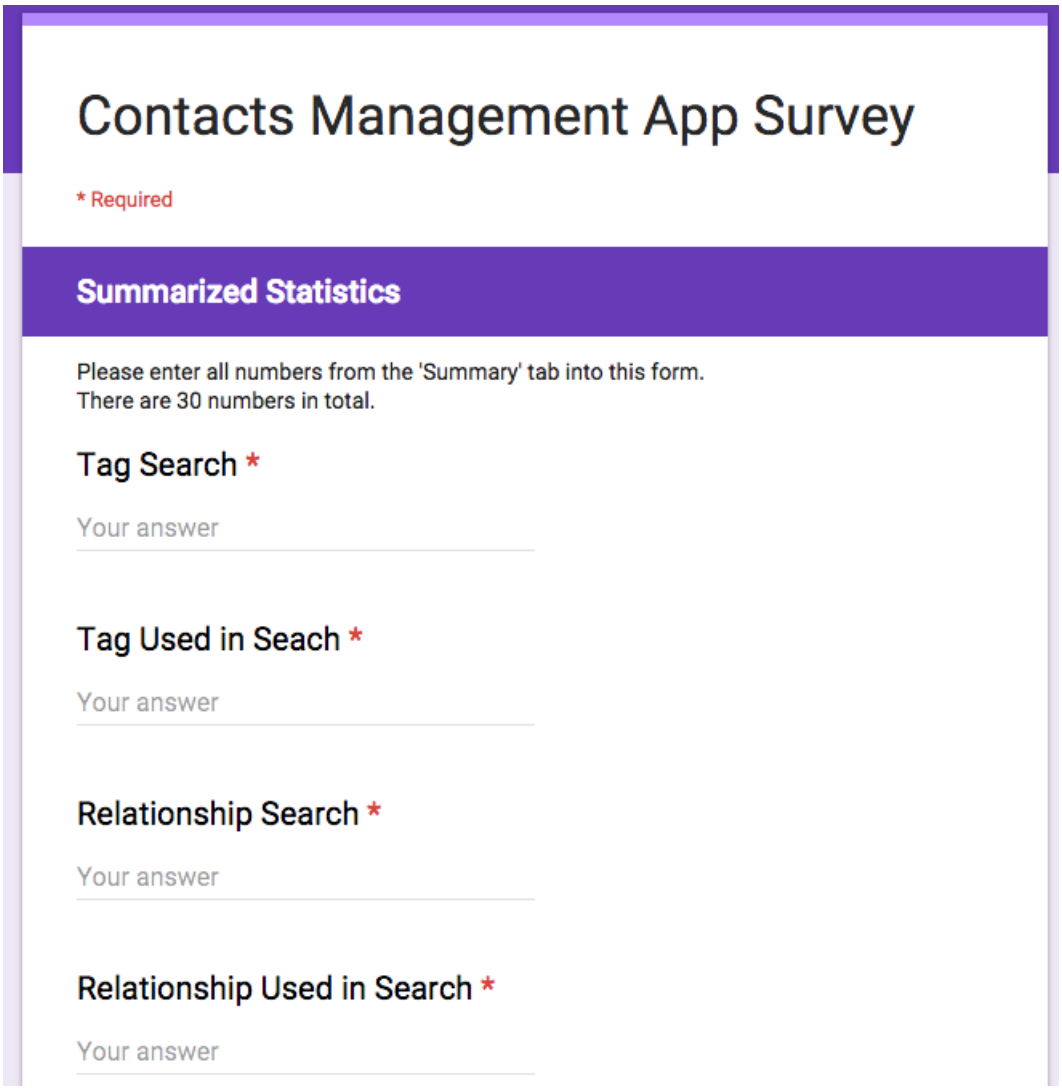

Figure 4.2: Graphy's Online Form

4.3 Results and Discussions

This section presents the results of the study. The section contains four parts. The first part *Tags and Relationships* 4.3.1 looks into the information participants store in Graphy which includes tags, relationships, and traditional information like names and phone numbers. The second part *Contacts Search and Relationship Traversal* 4.3.2 examines how participants consume the data they store in Graphy through searching and navigating the relationships network. The third part **?? ??** analyzes some general feedbacks we received from the users then discusses the strengths and weaknesses of the study. The last part summarizes what we achieve through this user study.

4.3.1 Tags and Relationships

The first few metrics of Graphy aim at evaluating how intensively the participants use the system. More specifically, the first metric that we calculate is Active Contacts Total - the total number of contacts which were viewed, edited or added during the study. All data from the twenty one respondents shows a moderately high number of Active Contacts Total. On average, a person uses approximately 33 contacts in 2 weeks, which means well over 2 contacts per day. The vast majority of the users have more than 15 active contacts as opposed to only a modest 14% of them have less than 15. The Active Contacts Total data can be seen in Table 4.1 and the group of Figures 4.3, 4.4, 4.5. From the data, we can infer that participants use Graphy quite a lot.

Table 4.1 also shows the other two metrics: Tags Total and Relationships Total which are the total number of tags and relationships respectively. Regarding tags, the average total number is almost 90. A significant proportion of participants, 38%, maintain more than 100 tags. Nobody has less than 20 tags and 2 people strikingly have more than 200 tags. Dividing the number of tags by the number of active contacts gives us the average of tags per contact which we call Tag Average. To be more specific, the Tag Average among all participants is roughly 2.73 tags per contact. The highest Tag Average is 3.91 and the lowest is 1.68. These are promising numbers. Generally, every user creates at least one tag for his/her contacts and on average almost 3 tags per contact. Because tags represent miscellaneous information users put into their contacts, the higher the numbers, the more miscellaneous information is created. From the collected data here, we can conclude that participants need about 3 pieces of miscellaneous information for every contact. Notably, the tags recorded here do not include Auto Added Tag (tags that the system automatically adds to a contact like *Date Created*, *Location Created*). In comparison with tags, the numbers of relationships are slightly smaller. On average, a respondent creates 55.52 relationships over the course of 14 days. Only 2 users, that is 9.5%, maintain more than 100 relationships. When it comes to Relationship Average, typically each contact has 1.63 relationships. There are three people maintaining the highest Relationship Averages at 2.02, 2.03, and 2.04. On the contrary, there is one person who has only 0.78 relationships per contact. Although the users tend to create fewer relationships than tags, 1.63 relationships per contact is still a very positive indicator.

Apart from calculating the total numbers of tags and relationships, we also recorded tag and relationship types. Table 4.2 shows how many types of tags and relationships were created during the experiment. The Average Tag Type Count is about 65. The person with the most tag types has 126 types while the person who creates the least types has 27. Another thing which stands out in this table is that there are two people who have more tag types than tags. They both have very small numbers of tag types which are 29 and 30. These unusual situations can be explained by two reasons. First, Graphy has 4 pre-defined tags which are hard-coded in the application: *Colleague*, *Important*, *Created Date*, *Created Location*. We create these pre-defined tags as examples to guide users in how to use tags. These 4 tags are added to the number of tag types so it is possible that the users get 4 extra types without actually using them. Second, every time a user creates a tag type, the type is stored in the database so the user can use it for multiple contacts. If the

Table 4.1: Total Numbers of Active Contacts, Tags, and Relationships

Active Contacts Total	Tags Total	Tags Average	Relationships Total	Relationships Average
46	180	3.913	94	2.0435
72	241	3.3472	146	2.0278
55	127	2.3091	67	1.2182
16	27	1.6875	30	1.875
50	101	2.02	87	1.74
9	22	2.4444	7	0.7778
8	29	3.625	12	1.5
49	165	3.3673	86	1.7551
20	48	2.4	39	1.95
18	69	3.8333	34	1.8889
32	68	2.125	45	1.4063
29	67	2.3103	33	1.1379
23	60	2.6087	29	1.2609
24	47	1.9583	38	1.5833
30	102	3.4	58	1.9333
63	207	3.2857	128	2.0317
8	26	3.25	10	1.25
52	111	2.1346	80	1.5385
41	85	2.0732	71	1.7317
22	41	1.8636	41	1.8636
17	56	3.2941	31	1.8235
Average:				
32.5714	**89.4762**	**2.7262**	**55.5238**	**1.6351**

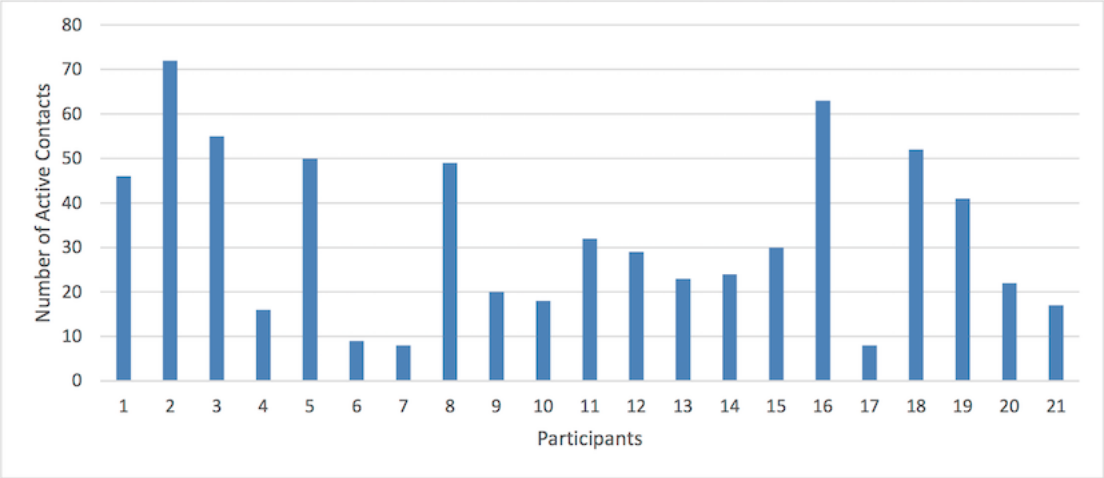

Figure 4.3: Number of Active Contacts

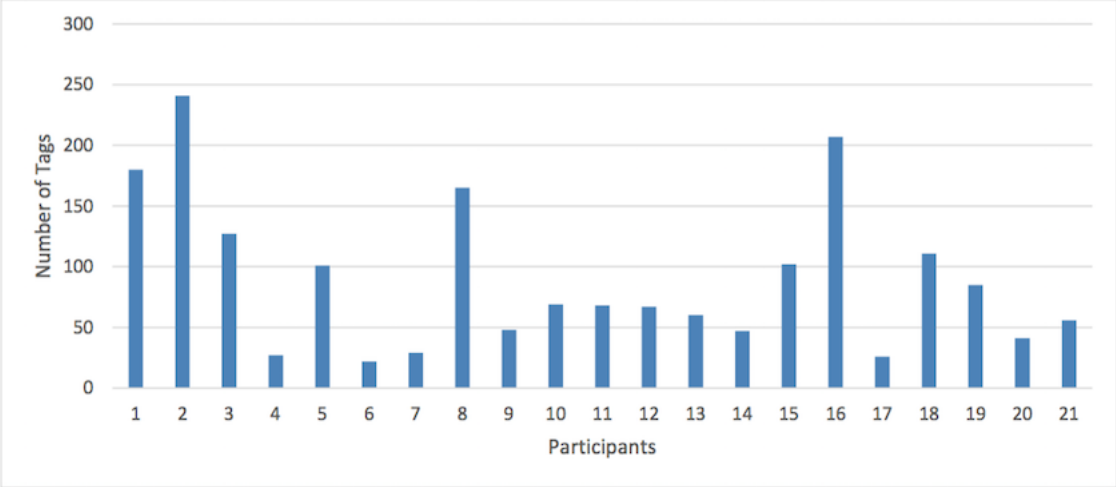

Figure 4.4: Number of Tags

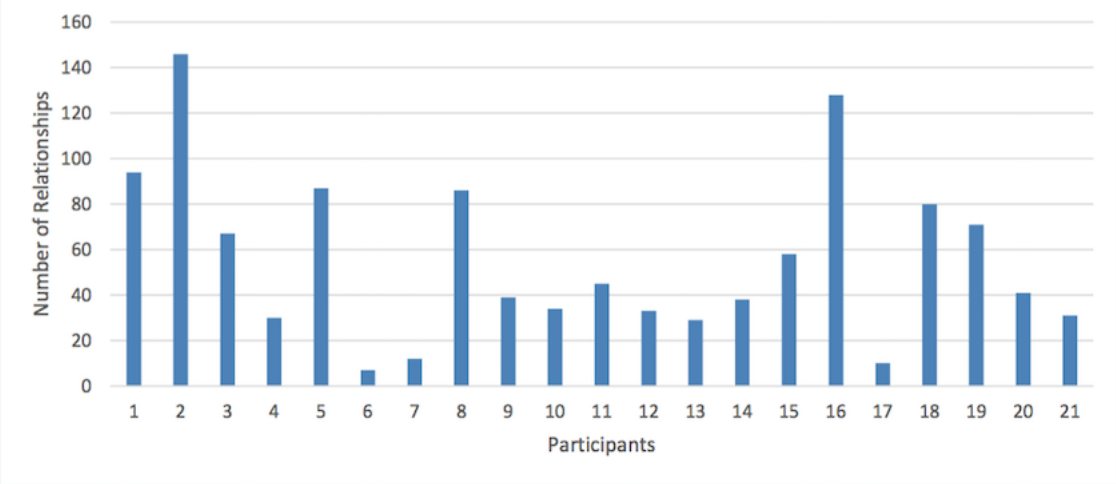

Figure 4.5: Number of Relationships

user creates some types then never uses them, or uses the types on a contact then deletes that contact, the total number of tag types will be increased by the amount of unused types. This fact reveals a limitation in our system design. Ideally, Graphy should have a clean up function which deletes unused types after a long period of time. To some extent, the ratio of the number of tags to the number of tag types implies how many times a tag is reused. From table 4.2, the average ratio is roughly 1.3 tags for each tag type. Noticeably, there is one participant who maintains a ratio of exactly 1. Overall, this *reused* ratio is fairly low. On the one hand, that means miscellaneous information is broad and needs many different tag types to cover. On the other hand, it exposes another limitation of the system: poor recommendation for tag/relationship types. At the moment, our application lets people enter new tag type names or pick an existing tag type from a list. A better solution should be giving suggestions based on the existing types as users enter type names. Moreover, Graphy is currently case-sensitive which in turn may be inflexible. For instance, it will consider these two names different types "Important" and "important". Therefore, making Graphy case-insensitive or equipping it with the capability of detecting similar words is quite important. These two improvements will be reserved for future work. In comparison to tag types, the number of relationship types is smaller by a large margin. The average number of relationship types is only 17, nearly 4 times smaller than the tag types. Concerning the relationships reused ratio, it is just almost three times as big as the ratio of tags. Notably, there is one respondent having the relationship reused ratio of 7.9 which is substantially high while the lowest relationship reused ratio is practically 2 relationships for each relationship type. These numbers suggest that relationship types are very well used, and people appear to have fewer kinds of relationship.

When it comes to measuring the impact of Graphy's new elements to a contact, we use another two metrics: Tag Weight Average and Relationship Weight Average. The way these metrics are calculated is as follows: on a contact we count the number of tag fields and the number of relationship fields then divide them by the number of all fields from which we get Tag Weight and Relationship Weight. The averages of Tag and Relationship Weight are then computed among all the contacts of a participant. The histograms of the two metrics are illustrated by Figure 4.6 and 4.7 and the detailed results can be found in Appendix A. As regards Tag Weight Average, each participant generally maintains a considerable high number at roughly 27%. Remarkably, the vast majority of participants (10 people) have Tag Weight Average between 31% and 35%. Nobody has less than 16% or more than 40% on this metric. It is worth noting that Auto Added Tags such as *Date Created* and *Location Created* are not included in the calculation. If we add these two simple tags into the formula, Tag Weight Average can be increased by a large margin. By comparison, Relationship Weight is a little lower than Tag Weight at approximately 17%. Only one respondent has Relationship Weight Average by less than 10% while the greatest proportion of respondents, 57% (12 people), keep this metric in the 16-20% range. There are four users with Tag Weight Average between 11% and 15%, at the same time another four users have the metric between 21% and 25%. These data are again very promising. On average, tags and relationships together serve almost half the information in a contact. The pie chart 4.8 displays more clearly that Graphy's newly introduced elements (tags and relationships) are practically as many as

Table 4.2: Tag and Relationship Types

Tag Total	Tag: Type Count	Tag: Total/Type	Relationship Total	Relationship: Type Count	Relationship: Total/Type
180	114	1.5789	94	23	4.087
241	106	2.2736	146	36	4.0556
127	126	1.0079	67	36	1.8611
27	27	1	30	14	2.1429
101	91	1.1099	87	11	7.9091
22	29	0.7586	7	4	1.75
29	30	0.9667	12	5	2.4
165	117	1.4103	86	27	3.1852
48	39	1.2308	39	15	2.6
69	52	1.3269	34	10	3.4
68	67	1.0149	45	23	1.9565
67	64	1.0469	33	18	1.8333
60	42	1.4286	29	15	1.9333
47	39	1.2051	38	17	2.2353
102	56	1.8214	58	16	3.625
207	94	2.2021	128	33	3.8788
26	20	1.3	10	5	2
111	104	1.0673	80	20	4
85	78	1.0897	71	10	7.1
41	39	1.0513	41	13	3.1538
56	45	1.2444	31	10	3.1
Average:					
89.4762	**65.6667**	**1.2922**	**55.5238**	**17.1905**	**3.2479**

traditional pieces of information (e.g. names, phone numbers, emails, etc.). From our experience, a contact typically contains 3 to 5 traditional fields: a first name and last name, one or two phone numbers, an email, and sometimes the organization that the contact belongs to (interestingly, the organization is often placed together with the names). Since traditional information takes up 56% of the total fields, the number of tag fields is between 2 and 3 while the number of relationship fields is 1 or 2. This observation perfectly matches with the Tag Average and Relationship Average data we have just discussed in the previous paragraph (which are 2.7 and 1.6 respectively). To conclude, the results show that tags and relationships constitute a very large part of a contact. They contain special pieces of information which is important to retain knowledge of users' contacts.

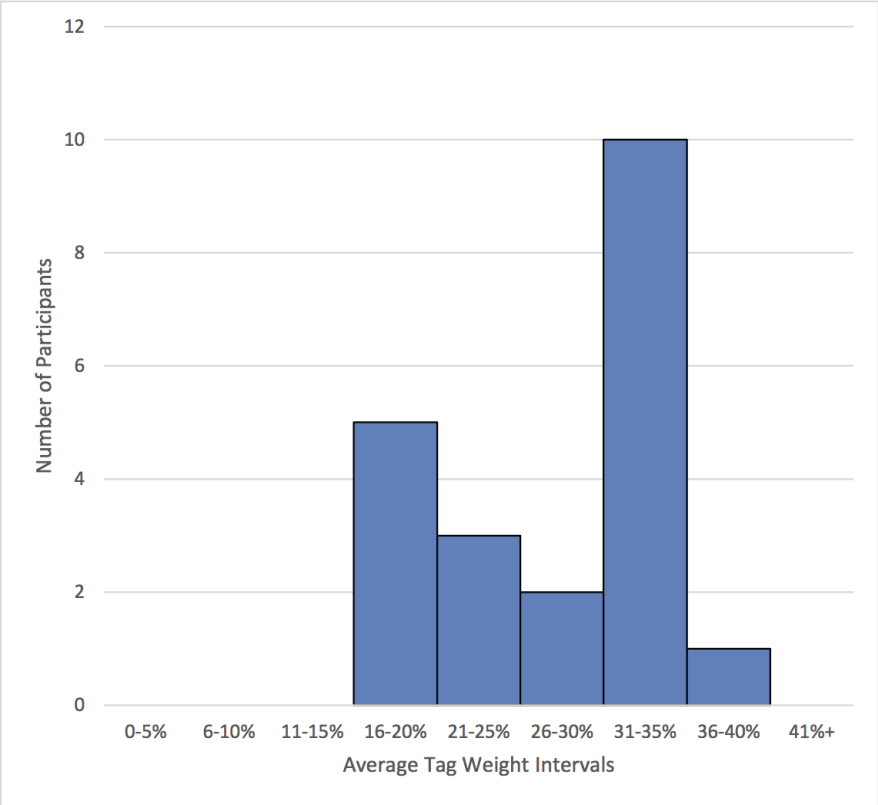

Figure 4.6: Average Tag Weight Distribution among Participants

4.3.2 Contacts Search and Relationship Traversal

The group of metrics we analyze in the previous section focuses on investigating how people store their contacts information in Graphy. In this section, we are going to explore the other group of metrics which examines the way people consume their stored information.

The first aspect to go into is how users search for contacts. In order to investigate this, we record

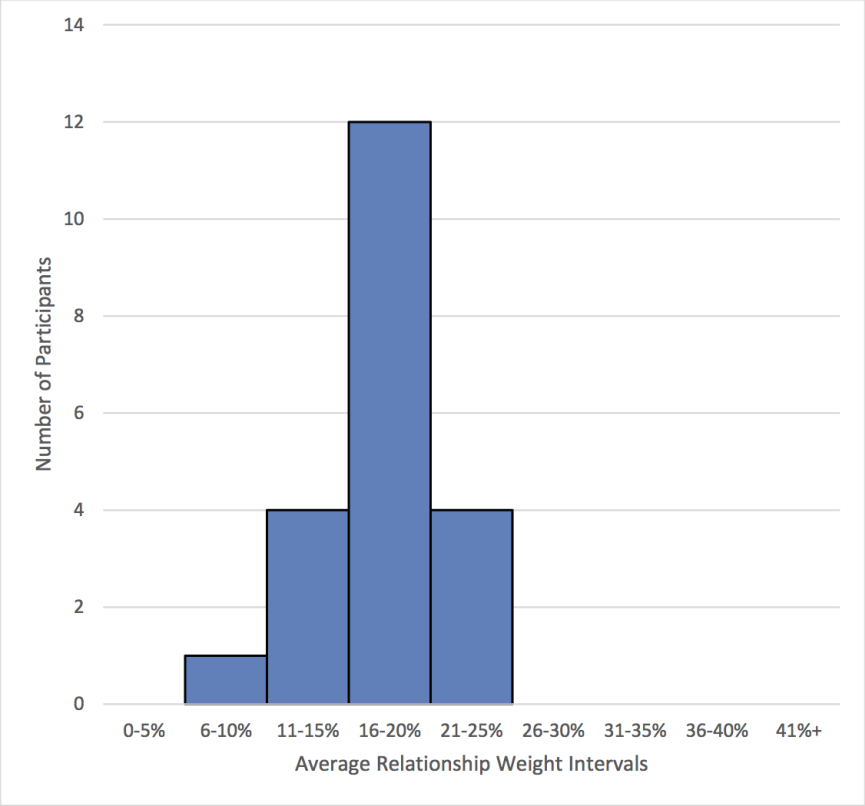

Figure 4.7: Average Relationship Weight Distribution among Participants

the number of search activities they perform during the experiment. It is important to note that we only count successful searches which yield at least one result. On average, a user looks up just over 32 contacts over the course of 14 days which equals to more than a search per day. This means the users use Graphy to retrieve information quite often. Particularly, one participant uses the search function 73 times while the two least active users use it 12 times. The detailed results of the study can be found in Figure 4.9. Among all the search activities, the first type of search we want to mention is the Traditional Search. A Traditional Search occurs when a user looks up contacts using *traditional information* like names, phone numbers, emails, organization, etc. This is the type of search people are currently performing in contacts management applications nowadays. The average of Traditional Searches is 8.67 times or approximately 27% of the total number of searches. The highest number of Traditional Searches is 19 (61%) while there are two users who do not use traditional information to search at all. The data here is surprisingly low. It seems participants are moving away from the traditional method of searching and utilize the newly introduced elements in Graphy to search. Regarding tags, the number of Tag Searches are strikingly high. Although tags take up only 27% information of a contact (Figure 4.8), they are the pieces of information participants use the most for searching. In general, a person uses tags to search more than 58% of the time. On the

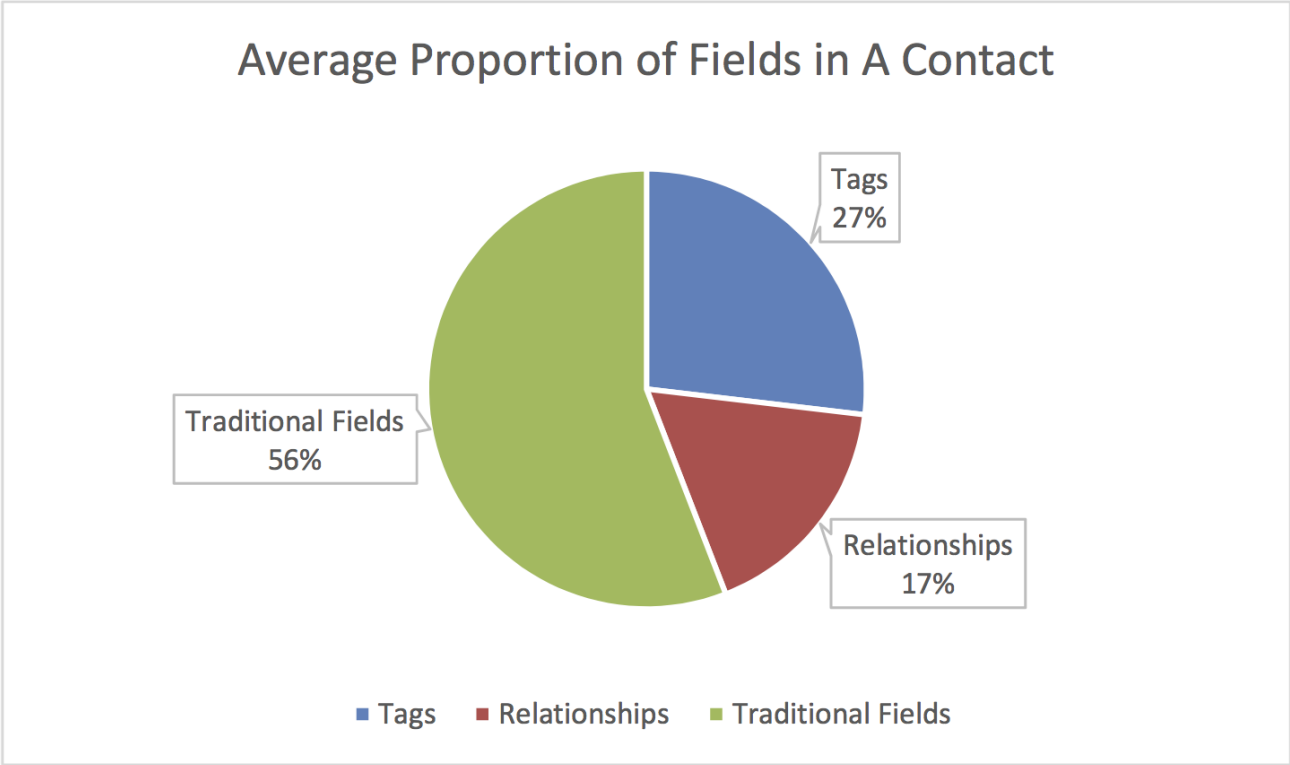

Figure 4.8: Average Proportion of Fields in A Contact

contrary, a modest 14.6% of the total searches is done via relationships. The average proportions of search types are illustrated clearly by Figure 4.10. Another outstanding point in the data is that one participant use Tag Searches 13 times in his/her total 14 searches but only use Traditional Search once and never use Relationship Search. There are actually two people who do not use Relationship Search in our study while the lowest Tag Search percentage is over 36%. In conclusion, the vast majority of searches are accomplished through tags which are above two times as many as Traditional Searches while Relationship Searches are the lowest among the 3 types.

The second aspect we investigate in the experiment is the way people explore the relationships network. In fact, traversing the relationships network can be considered a special form of searching for information. Every time a user travels from one contact to another through the relationship connection between them we increase the tracking variable *Relationship Navigations* by one. By contrast to the low number of Relationship Searches, the average of Relationship Navigations is significantly high: at 19, almost two thirds of the total number of searches. In other words, each participant uses this feature nearly 1.5 times everyday. Especially, one participant navigates his/her relationships network 62 times during the study. According to Figure 4.11, the greatest proportion of users, that is 9 (43%), traverse the network between 11 and 20 times, and only a minority of them (23%) uses the functionality less than 10 times. From the data, we can deduce that relationships are not often used directly for searching. Instead, they turn out to be valuable when the users

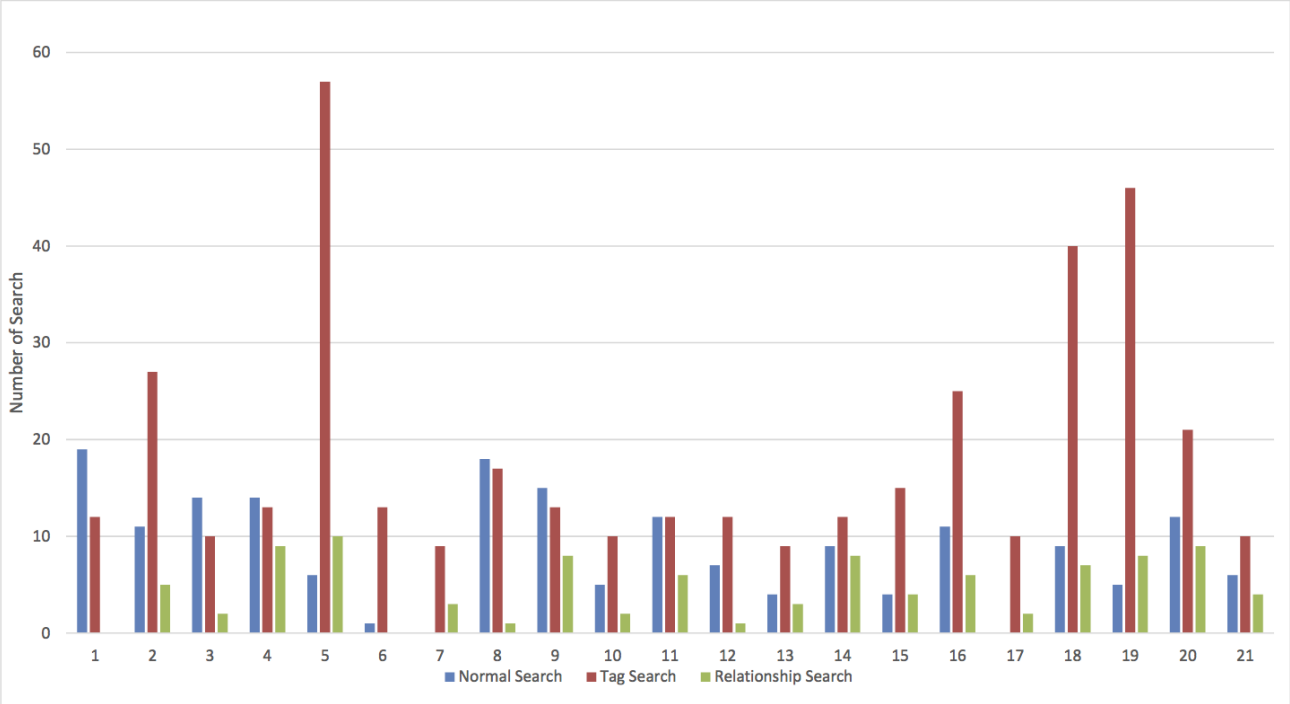

Figure 4.9: Total Numbers of Searches

want to find a contact through the information of another contact. This fact is actually understandable. Directly searching for a relationship can only answer a special type of requests, for example "Find me all contacts who have a daughter!". This type of requests is definitely rare in real life scenarios so direct relationships searches are few. On the other hand, relationships navigations can help users to find a contact when the only thing they remember is that contact is a friend/colleague/relative of someone. This type of request is surely more common. Moreover, navigating the relationships can also be used to explore the whole circle of friends. Therefore, it is frequently used.

The last aspect of the contact searches we examine is the number of tags and relationships used in each search. Regarding tags, the average number of tags used in all the searches is just over 19. Since the average total number of tag searches is 18.7, the average number of tags per search is practically 1. It means the users often use only 1 tag for every search. Only a minority of participants, 23%, has Tags Per Search higher than 1. In a similar fashion, participants use exactly 1 relationship in each relationship search. This result reveals that people tend to be *lazy* while searching. They only enter one criterion then pursue the search. Perhaps they like scrolling the list of search results to find the desired contact more than typing more search criteria to filter the list further. It is totally reasonable because we all know people do not like typing a lot. Another possibility is that users usually can only recall the most distinct characteristic of the contact they want to find. These conclusions lead us to an improvement we can add to Graphy in a future version: recommending relevant tags while users type search criteria.

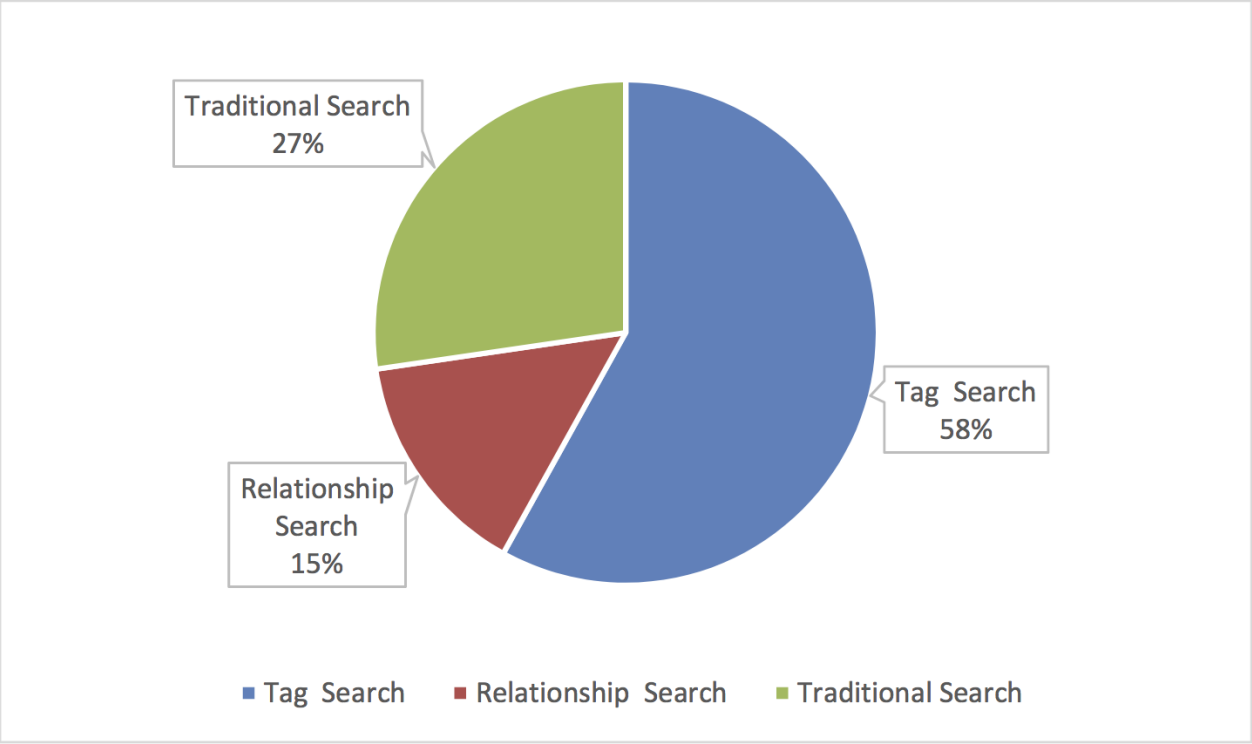

Figure 4.10: Average Proportion of Searches

To sum up, we have just investigated the way participants search for contacts and navigate their relationships networks. The results are very promising. People often use the newly introduced elements in Graphy to look for information. Despite taking up 56% of a contact, traditional information like names or organizations only contribute a mere 27% of the total search while tags outstrip it and account for 58% of the searches. As for relationships, although they are rarely used in direct searches, users utilize them to explore the relationships connections and to find a contact when it is connected with someone more memorable. However, the data can be a little in favor of tags and relationships because participants feel that they are using a special application with exclusive features so they try the features, instead of actually using them. But even so, we still think tags and relationships have confirmed their values. In the future, we will conduct another study which has a timeline to see if people increasingly use tags and relationships instead of traditional information over time.

4.4 Summary

According to the analyses and discussions above, we can say that the user study has confirmed our 3 hypotheses. As regards the first hypothesis, Graphy is not only capable of finding contacts by miscellaneous

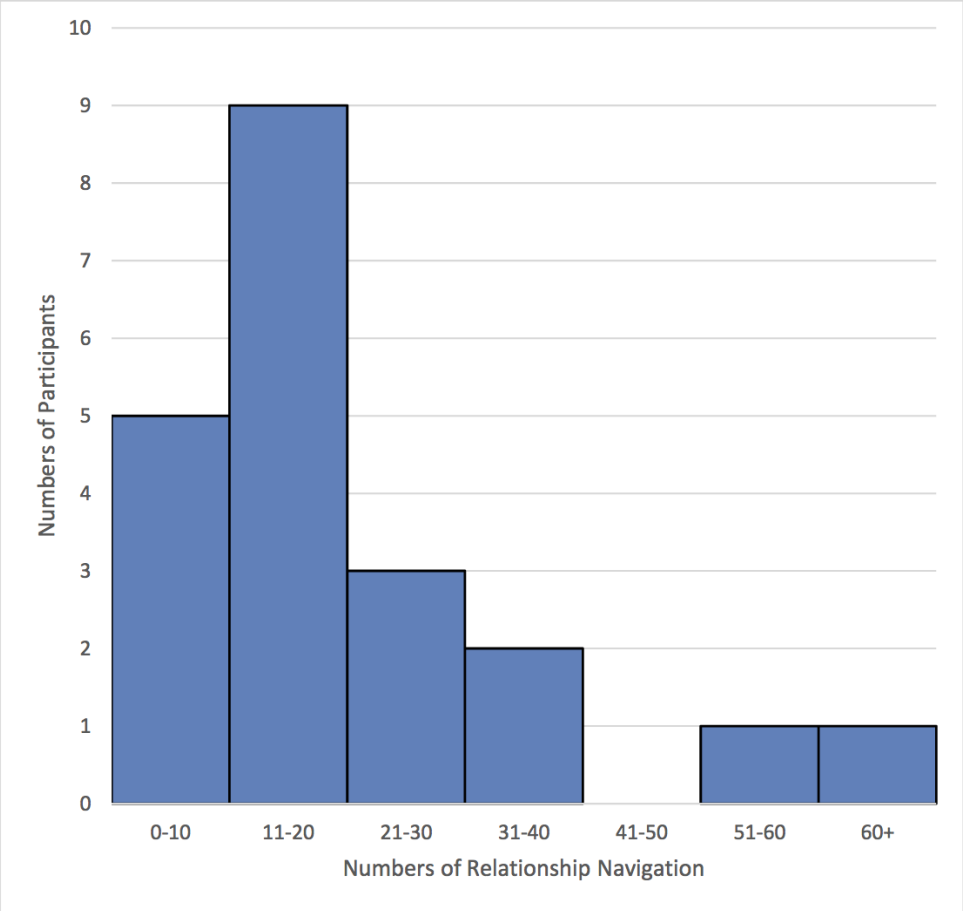

Figure 4.11: Relationship Navigations Distribution among Participants

information, but it also does it very well. Searching by tags ranks first among all searches with a striking percentage at 58% while traditional searches take up only 27%. It is also worth noting that to the best of our knowledge Graphy is the only application that can perform this type of searching while popular contacts management applications on iOS or Android can only look up a set of pre-defined fields. As for the second hypothesis, our system clearly allows people to retain knowledge of their contacts through the tags system as well as traditional fields such as names, phone numbers, and emails. According to the survey data, participants maintain a significant amount of miscellaneous information via tags. To be more specific, the average number of tags among all participants is about 90, the average number of tags per contact is 2.72, and tags account for nearly 30% of the information in every contact. Moreover, the fact that participants frequently use the tags they store to search also confirms that tags are useful and truly efficient in retaining knowledge of contacts. Concerning the third hypothesis, Graphy supports users in establishing relationships between their contacts. Additionally, by clicking on the relationships users can traverse the relationships network easily. The high number of relationships and relationship navigations affirms the accessibility of these features.

Overall, the user study shows that our new functionalities are helpful and have many of potentials. In fact, the functionalities actually help users solve some existing problems they have in real life. For instance, two participants tell us that almost every contact's first names in their list are made up of names and some other info thus the first names become really long, one participant often has to scroll all the list to find a contact, and to our surprise he does not find that contact by the name (which he has forgotten) but by the event happening when they met. Furthermore, Graphy reveals an innovative behavior from the participants: many of them start creating a *Self* contact in order to connect with other contacts in the relationships network. This special contact turns into a central point which helps the users connect different parts of the network then makes navigations much easier.

On the other hand, the user study still has a few limitations. First, the number of participants is small and our only requirements to a participant are having a smart phone and being familiar with it. Obviously, a larger group of users with more diversity in cultures, educations, professions, and technology familiarities will be much more valuable to the experiment. Second, the duration of the study is fairly short. If the participants have more chance to use the application we may discover more information about their behaviors. Although Graphy is simple and has a shallow learning curve, people still need some time to get used to it. Third, the experiment is conducted in an uncontrolled environment. Despite the fact that most of the users' actions are recorded, we still could not capture the whole interactions between the users and the application. If the study is carried out in a more controlled environment like a lab with cameras or assisted personnels, we will understand the users better. However, this type of controlled study is costly, more limited in duration, and it will compromise users' privacy. Finally, the last limitation of our study is that we did not ask participants to answer any questionnaire. We decided not to require people to fill in a templated questionnaire since we wanted them to only focus on using the application. Nevertheless, with a good questionnaire we can know how people enjoy the system, their difficulties, and we can get more feedbacks and suggestions from them. In our future work, we will tackle these limitations and improve Graphy to capture more users' behaviors.

Chapter 5

Experiment and Evaluation - System Performance

5.1 Goals

This experiment is just a proof of concept to show that Graphy's data models for tags and relationships can be exchanged and synchronized between mobile clients like other Contacts application with similar performance. The goals of the experiment are as follows:

- Clients can push contact data to backend server.
- Clients can pull contact data from backend server.
- Clients can synchronize their contact data with each others (through the help of the backend server).
- Clients can resolve contact data conflicts between each others (through the help of the backend server).

5.2 Experimental Setups

5.2.1 Overview

The experimental system basically contains three parts: the clients, the backend server, and the communication between them. Everything runs on a Macbook Pro 2012 with the specifications listed in Table 5.1. The Macbook provides a Unix-like environment where us can run everything we need easily. However, comparing with a real-world, production system, our setup is slower by a huge margin. To be more specific, we use multi-threading programming in order to simulate multiple devices. For communication, we just run the clients and the backed server on the same local host. This method eliminates the delay of the actual Internet connection so we can focus on examining the synchronization process. Even though we can find better equipments, we decide to run our tests on this slow system, if the experiment yields good results, it means it will work even better on a more powerful environment.

5.2.2 Clients

The actual fully-featured mobile clients are written in C# on the Xamarin platform. However, in this experiment to reduce the manual tasks to a minimum we re-write the clients into multiple threads, each thread

Table 5.1: System's Specifications

Processor	2.5GHz dual-core Intel Core i5 (Turbo Boost up to 3.1GHz) with 3MB L3 cache
Memory	8GB of 1600MHz DDR3 memory
Storage	500GB 5400-rpm hard drive
Operating System	OSX Yosemite v.10.10

represents one client. We also discard all the unnecessary code like the user interface and other unrelated functions. The experiment code of clients can be accessed at: `https://github.com/NamXH/GraphyClient`, and the fully-functional mobile clients' code can be accessed at: `https://github.com/NamXH/GraphyPCL`. As a result, we take full controls of the client threads and we can trigger the synchronization process at any specific moment precisely. Moreover, using the asynchronous programming model with Async and Await in C# we can run many client threads in parallel to simulate multiple devices being used at the same time. Besides, since we only test the performance of the data synchronization which operates on each user separatedly so for fast prototyping we do not implement authentication in the experiment.

On the other hand, the databases of the clients are carefully implemented for every thread. We also use SQLite which is the same technology as the mobile devices. The details of the database schema can be found in Appendix B. Regarding the Sync Queue, each modification on the database (creating/updating/deleting a business record) is mapped to a row in the Sync Queue table. For instance, a creation is be mapped to a POST request, an update is mapped to a PUT request, and a deletion is mapped to a DELETE request. Figure 5.1 shows some examples of the Sync Queue table.

	Id	Verb	ResourceEndpoint	ResourceId
55	8e608ab5-24fd-4f6b-a62a-7cec926a468c	Put	contacts	23375079-8659-4c0c-9fe6-01f69aff9dfa
56	db5c21ef-9946-457d-a4d9-c508ee375cf0	Post	tags	41701315-f588-468d-a974-34e95e10ed91
57	e0738623-f1e1-4ac0-b99c-69ac6d87301c	Post	contact_tag_maps	d77206b7-c84d-4fb3-a734-99e4ec052828
58	e1bd30c1-928a-451e-b255-7d9ad3d6eb10	Post	relationship_types	18c9066e-b3ec-4af5-859a-29eec20e1841
59	53fa6a4d-be94-44fb-bf70-6a6072607f01	Post	relationships	0ab9d447-f03a-4612-b03a-5e7a240013ea
60	2a655d92-e0e5-44e5-9f93-65bea3ab0800	Put	contacts	cc363bba-fc66-458e-8f4d-aeae3d48a92e
61	5f0ee5fa-68a4-4fcd-abc7-08ad0693eb48	Post	tags	3f7eca8d-5a4c-4894-aa81-e09ab82ac88b
62	45c70dc9-d85c-439c-854e-dd22581286b0	Post	contact_tag_maps	a74db34c-8a4e-4972-9426-b11dfa8510ea
63	32f1e00b-511c-42f1-95b4-49e4ba3063be	Post	relationship_types	6ad29638-403e-433b-b4ce-9320f62fd555
64	69bede11-1316-4df0-a14d-972e04ec7804	Post	relationships	193eae3e-3195-4312-93b2-2d2a46206ca0
65	004782ba-229f-4df1-ae6a-5a6fd0d8a109	Put	phone_numbers	90efd42b-0d62-47bd-8e37-d066f816ff22
66	512f9bdb-d5fc-4196-85a8-02ed612a227f	Put	phone_numbers	952581d2-3391-4a71-9c54-64301936668f

Figure 5.1: Sync Queue Table

5.2.3 Server

The backend server is written in Python using the Django framework. Its code can be accessed at: `https://github.com/NamXH/GraphyBackend`. Figure 5.2 presents some examples of the database models written for the server. We run the Python code on the Gunicorn web server with 5 workers since our processor has 4 cores. It is also worth mentioning that the server's database is implemented using PostgreSQL - a high performance, open source, relational database. Python and Django have incredible support for PostgreSQL so the setup is very simple. Moreover, we can use the Python object-relational mapper to access PostgreSQL instead of writing raw SQL queries.

```
18  class Tag(models.Model):
19      Id = models.UUIDField(primary_key=True, default=uuid.uuid4)
20      Name = models.CharField(max_length=255, null=True, blank=True)
21      LastModified = models.DateTimeField(null=True, blank=True)
22      IsDeleted = models.BooleanField(default=False)

32  class ContactTagMap(models.Model):
33      Id = models.UUIDField(primary_key=True, default=uuid.uuid4)
34      Detail = models.CharField(max_length=255, null=True, blank=True)
35      ContactId = models.ForeignKey(Contact, null=True, blank=True)
36      TagId = models.ForeignKey(Tag, null=True, blank=True)
37      LastModified = models.DateTimeField(null=True, blank=True)
38      IsDeleted = models.BooleanField(default=False)
```

```
10  class TagSerializer(serializers.ModelSerializer):
11      Id = serializers.UUIDField()
12      class Meta:
13          model = Tag
14          fields = ('Id', 'Name', 'LastModified', 'IsDeleted')
```

Figure 5.2: Backend Server Implementation

5.2.4 Communication

The communication between the clients and the server is achieved using RESTful HTTP requests. The server provides an API which contains 11 endpoints, each endpoint is corresponded to a table in the database such as contacts, phone numbers, tags, relationships. An endpoint responds to the HTTP requests sent by clients differently to perform the CRUD (Create-Read-Update-Delete) operations on the database. Furthermore, to simulate the intermittent nature of the Internet, we sometimes perform an interruption to the synchronization process. Besides, while many mobile clients are being used by the users, there can be situations that some conflicts occur. Normally, the users have to manually resolve the conflicts. However, to reduce the complexity,

we create a time stamp of the last modification on an entry and use it to decide which version to keep (i.e., we keep the latest version).

5.3 Scenarios and Results

In order to test whether our system achieves the goals, we ran it through different scenarios and evaluated the results. The default data set used in every scenario consisted of 200 contacts, each contact had 3 traditional fields: first name, phone number, and email. Additionally, there were also 100 tags attached to 100 contacts, and 100 relationships which connected 100 pairs of contacts. The details of the scenarios will be discussed in the following sections.

Scenario 1: Creation

The first scenario aimed at testing the first goal: *Clients can push contact data to backend server*. Therefore, we created an initial database with the default data set in the client then started the synchronization process. This scenario tried to simulate the situation when a person uses Graphy for the first time in which he/she imports and creates a few contacts then starts the synchronization process when the Internet connection is available. Each creation or importation in Graphy is associated with a row in the database, then each row has a corresponding POST operation in the Sync Queue. It is important to note that before the first synchronization the users may also modify or delete some of their new entries. As a result, the Sync Queue may contain PUT and DELETE operations. However, in Graphy's algorithm instead of creating new PUT or DELETE operations of a new entry we inspect the actions and just modify the corresponding POST operation so there is only one POST operation for every newly created entry. Therefore, the Sync Queue for the initial database consisted of 1000 POST operations which represent all the entries in seven tables: Contacts, Phone Numbers, Emails, Tags, Contact-Tag Maps, Relationship Types, and Relationships.

After running the scenario, our backend server had the same data as the client which was a success indicator. The results of the scenario are illustrated in Table 5.2 and Figure 5.3. The results are very promising when the average time to completion of 10 tries is only 9.79 seconds. Particularly, the vast majority of the outcomes are less than 10 seconds. Furthermore, the time to completion will be significantly faster if the scenario is operated in high performance, dedicated servers. In short, the first goal of the system is achieved after examining the first scenario. The speed of the system is also acceptable because this scenario does not happen very often in real life.

Scenario 2: Modification

The second scenario tried to simulate another situation in real life: a user use Graphy in the offline mode (without Internet connection). While using the application in the offline mode, the user can still perform all the actions normally. The actions are stored in the local database then synchronized with the backend server

Table 5.2: Time to Completion of The Scenarios (seconds)

Creation/Modification	Retrieval	Gmail Retrieval
9.59	4.03	18.47
9.75	4.58	20.77
9.85	5.03	19.73
9.49	4.30	17.92
9.57	4.63	21.81
9.63	4.99	19.89
10.11	5.00	18.32
9.59	4.21	20.59
10.50	4.88	19.29
9.79	4.90	20.07
Average:		
9.79	**4.66**	**19.69**

when the Internet is available. With this in mind, we made various changes to the data of the client in scenario 1 in the offline mode, then we triggered the synchronization process and waited for the data to be uploaded to the backend server. The types of changes we made included: creating new records, updating/deleting current records, and creating new tags and relationships for the existing contacts. Notably, when deleting a record, we had to delete all other related records and their operations in the Sync Queue. For example, when deleting a contact, we had to delete its phone numbers, emails, tags, relationships as well. In total, there were 90 synchronization operations needed to be transfered to the server in this scenario. We tested the scenario for 10 times and the synchronization process happened almost instantly every time (less than 1 second). Therefore, we can conclude that the modification performance is acceptable and strengthens the first goal: *Clients can push contact data to backend server.*

Scenario 3: Retrieval

Nowadays, one person may own more than one device. In this scenario, we tried to simulate the case when an user starts using Graphy on another device. The new device has to connect with the server to retrieve existing data of the user to its local database. Consequently, we created an empty database in a new client then triggered the synchronization process so the new client can fetch existing data from the server. The existing data was actually the information uploaded by the first client from scenario 1 and 2. To be specific, the new client sent GET requests to all endpoints of the server's API.Each endpoint corresponded to a table in the server's database. It is important to note that we had to synchronize the table without foreign key first. For instance, the Contacts table was synchronized before the Phone Numbers table because the Phone

Numbers table has a foreign key which referred to the primary key of the Contacts table. To sum up, the new client received 1000 entries from the server.

We tested the scenario 10 times and the outcomes are reported in Table 5.2 and Figure 5.3. Furthermore, we ran another experiment on the Gmail Contacts service to compare its performance with our system's. The Gmail experiment was set up similarly to scenario 3. First, we created 200 contacts on Gmail then we performed the synchronization process on a new device to fetch these 200 contacts. The device we used was the Apple iPad 4 which has the Apple A6X chipset, Dual-core 1.4 GHz CPU, and 1 GB of RAM. The device operated on a Wifi network which has about 6 Mbps download speed and 1 Mbps upload speed. The Gmail experiment's results are also included in Table 5.2 and Figure 5.3. From the table, we can see that the average retrieval time of Graphy is 4.66 while the retrieval time of Gmail is substantially higher at 19.69. The reason for Gmail being slower is it operated on the actual Internet connection instead of on a LAN environment like Graphy. However, as we have discussed previously, our system's hardware is far weaker than Gmail's servers. To conclude, we cannot determine that our system runs faster or slower than Gmail but we can say that it runs at an acceptable speed. Moreover, this scenario does not happen frequently, it only occurs when the user adds a new device or refresh his/her old ones. In summary, the results of this scenario has confirmed the second goal: *Clients can pull contact data from backend server.*

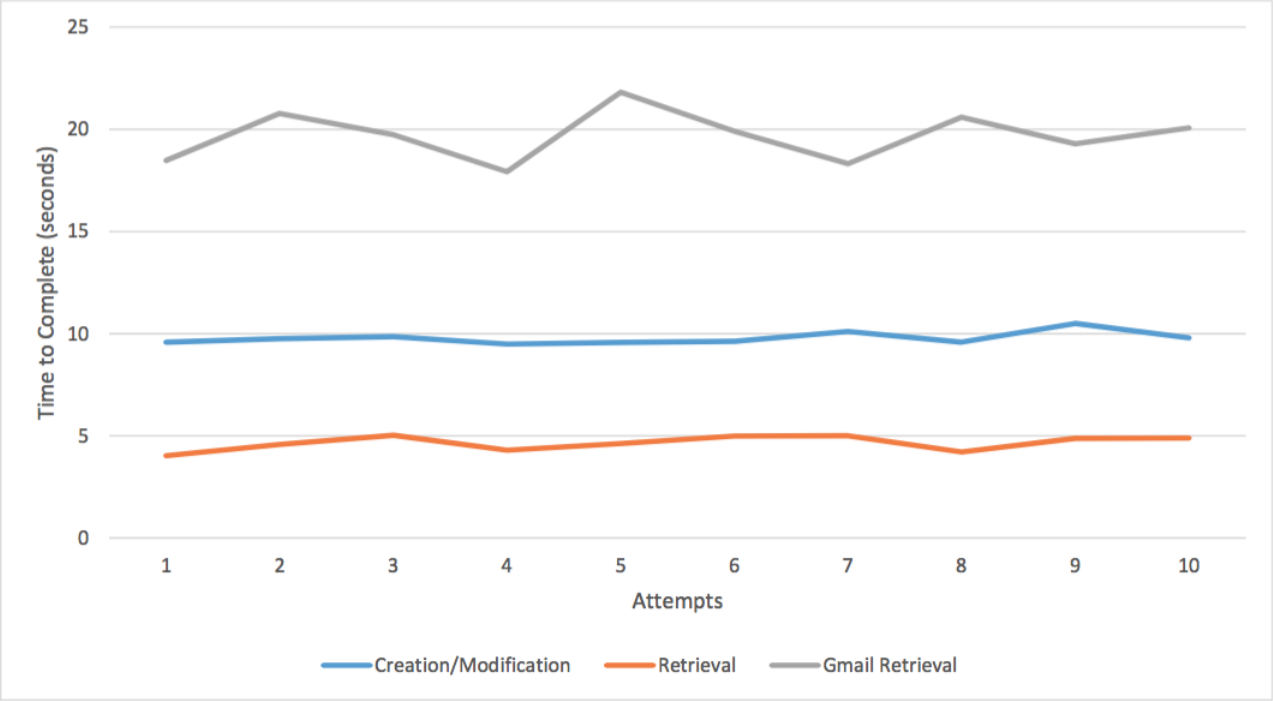

Figure 5.3: Time to Completion of The Scenarios

Scenario 4: Synchronization

This scenario addressed the third goal *Clients can synchronize their contact data with each others* and the real life scenario when a person uses more than one device simultaneously. While using multiple devices at the same time, changes on one device must be propagated smoothly to other devices. In order to simulate this scenario, we used the new client in scenario 3 (i.e. client 2) and made changes to its local database. The changes were similar to the ones in scenario 2 which resulted in 90 operations in the Sync Queue. After making changes, we triggered the synchronization process on this client so the data was uploaded to the backend server. Eventually, we took the first client in scenario 2 (i.e. client 1) and started the synchronization process. We tested the scenario 10 times and the results were very positive. The database on client 1 became identical with the database on client 2, which means the synchronization algorithm worked successfully. Moreover, the process happened almost instantly (less than 1 second) in all the tests.

Scenario 5: Conflicts Resolution

In this last scenario, we wanted to tackle the final goal: *Clients can resolve contact data conflicts between each others*. When a user uses many devices one at a time, it is possible that they mistakenly make different changes on the same entry multiple times on different devices. When this happens, we have a conflict to resolve. To simulate this situation, we made changes on a set of 20 entries in the database of client 1. At the same time, on client 2 we independently modified the same set of entries with different contents. After that, we triggered the synchronization function on client 1. While client 1 was synchronizing, we created an interruption so client 1 only synchronized half of its operations to the backend server. At that moment, we triggered client 2's synchronization process and let it finish completely. Therefore, at that point the server had received half of the changes from client 1 and all the changes from client 2. Eventually, we continued the synchronization process on client 1 and let it deal with the conflicts caused by client 2. The result was: client 1 resolved the conflicts and its database became identical with client 2 (client 2 made the changes at a later time compared with client 1 so its modifications took over client 1's modifications). We carried out the experiment 10 times and the resolution process always happened immediately. Therefore, we conclude that our system has achieved its final goal of being able to resolve data conflicts between devices.

5.4 Conclusion

To sum up, according to the results of the experiment we can say that our system has achieved its goals. The data models for tags and relationships can be synchronized successfully using our algorithm. Moreover, in spite of running on a mock system with very limited computing power, the synchronization process managed to complete all the scenarios in a surprisingly short time. However, we emphasize again that our work does not focus on building the best system or creating the fastest synchronization algorithm. Therefore, the system can be optimized further when we want to deploy it to a larger environment in the future. For example,

we can implement opportunistic locking strictly on the server's database to make sure the select and update functions happen in one transaction.

Chapter 6

Conclusion

6.1 Summary and Expected Contributions

In this research, we proposed a new model for the contacts management applications which will achieve three goals:

- Introduce a new way to look up contacts by using their miscellaneous information.
- Efficiently retain knowledge of contacts by using a tag system.
- Enable users to establish relationships between their contacts then traverse and explore the relationships with ease.

By achieving these goals, our model helps its users to accomplish new tasks which are not currently supported by modern Contacts applications. It is expected to provide users a more powerful tool to manage their contacts and also easy to use. As a result, users can search for their contacts faster and recall contact information easier. Moreover, the novel "reversed social network" proposed by the research is expected to open a new research direction in exploring and utilizing the relationships between contacts.

6.2 Timeline

The expected timeline for this research is as follows:

- Complete implementation, start distributing the prototype to users: June 30
- Collect users' feedback, start analyzing users' data: July 15
- Finish writing thesis, ready for the defense: July 31

6.3 Future Work

Due to the time constraint, there are several areas our research has not been able to explore, namely:

- Contact relationships suggestions: In a future extension, our prototype could be able to suggest relationships to users instead of letting them manually identify them. There are many possible directions could be used for suggestions such as examining area codes in the phone numbers, extracting information from the device's mailbox, analyzing existing mutual connections.
- Unify with online social networks: Our "reversed social network" can be connected to online social networks like Facebook, LinkedIn. As a result, we can pull updated information from actual social networks into our application.
- Network visualization: Visualizing the "reversed social network" could provide users benefits like assisting them in having a better overview and insight of their circle of friends.
- Ubiquitous computing: Our architecture can be extended to support operations on more computing devices like smart watches, desktops and the web.

References

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

[2] Cisco 2014 annual report.

[3] Delicious, http://delicious.com/.

[4] Facebook, http://www.facebook.com/.

[5] Flickr, http://www.flickr.com/.

[6] Fring, http://www.fring.com/.

[7] Gmail, http://gmail.com/.

[8] Linkedin, http://www.linkedin.com/.

[9] Mysqlicious, https://code.google.com/p/mysqlicious/.

[10] Os x 10.9 tags, http://support.apple.com/kb/ht5839.

[11] Scuttle, http://sourceforge.net/projects/scuttle/.

[12] Skype, http://www.skype.com/.

[13] Snapchat, http://www.snapchat.com/.

[14] Stackoverflow, http://stackoverflow.com/.

[15] Tags: Database schemas, http://tagging.pui.ch/post/37027745720/tags-database-schemas.

[16] Tagsystems: Performance tests, http://tagging.pui.ch/post/37027746608/tagsystems-performance-testsg/.

[17] Toxi, http://toxi.co.uk/.

[18] Twitter, http://twitter.com/.

[19] Viber, http://www.viber.com/.

[20] Whatsapp, http://www.whatsapp.com/.

[21] Wordpress, http://wordpress.org/.

[22] Youtube, http://www.youtube.com/.

[23] Gregory D Abowd, Anind K Dey, Peter J Brown, Nigel Davies, Mark Smith, and Pete Steggles. Towards a better understanding of context and context-awareness. In *Handheld and ubiquitous computing*, pages 304–307. Springer, 1999.

[24] Heikki Ailisto, Petteri Alahuhta, Ville Haataja, Vesa Kyllönen, and Mikko Lindholm. Structuring context aware applications: Five-layer model and example case. In *Proceedings of the Workshop on Concepts and Models for Ubiquitous Computing*, pages 1–5, 2002.

[25] Morgan Ames and Mor Naaman. Why we tag: motivations for annotation in mobile and online media. In *Proceedings of the SIGCHI conference on Human factors in computing systems*, pages 971–980. ACM, 2007.

[26] Matthias Baldauf, Schahram Dustdar, and Florian Rosenberg. A survey on context-aware systems. *International Journal of Ad Hoc and Ubiquitous Computing*, 2(4):263–277, 2007.

[27] Danah Boyd. Friends, friendsters, and myspace top 8: Writing community into being on social network sites. 2006.

[28] Danah Boyd. Why youth (heart) social network sites: The role of networked publics in teenage social life. *MacArthur foundation series on digital learning–Youth, identity, and digital media volume*, pages 119–142, 2007.

[29] Danah M Boyd and Nicole B Ellison. Social network sites: Definition, history, and scholarship. *Engineering Management Review, IEEE*, 38(3):16–31, 2010.

[30] Jay Budzik and Kristian J Hammond. User interactions with everyday applications as context for just-in-time information access. In *Proceedings of the 5th international conference on intelligent user interfaces*, pages 44–51. ACM, 2000.

[31] Ciro Cattuto, Vittorio Loreto, and Luciano Pietronero. Semiotic dynamics and collaborative tagging. *Proceedings of the National Academy of Sciences*, 104(5):1461–1464, 2007.

[32] Harry Chen, Tim Finin, and Anupam Joshi. An ontology for context-aware pervasive computing environments. *The Knowledge Engineering Review*, 18(03):197–207, 2003.

[33] Anind K Dey. Context-aware computing: The cyberdesk project. In *Proceedings of the AAAI 1998 Spring Symposium on Intelligent Environments*, pages 51–54, 1998.

[34] Margery A Eldridge, Philip J Barnard, and Debra A Bekerian. Autobiographical memory and daily schemas at work. *Memory*, 2(1):51–74, 1994.

[35] Nicole B Ellison, Charles Steinfield, and Cliff Lampe. The benefits of facebook âĂIJfriends:âĂİ social capital and college studentsâĂŹ use of online social network sites. *Journal of Computer-Mediated Communication*, 12(4):1143–1168, 2007.

[36] Lev Finkelstein, Evgeniy Gabrilovich, Yossi Matias, Ehud Rivlin, Zach Solan, Gadi Wolfman, and Eytan Ruppin. Placing search in context: The concept revisited. In *Proceedings of the 10th international conference on World Wide Web*, pages 406–414. ACM, 2001.

[37] Gartner. Predicts 2014: Apps, personal cloud and data analytics will drive new consumer interactions. Technical report, Gartner, 2014.

[38] Scott A Golder and Bernardo A Huberman. Usage patterns of collaborative tagging systems. *Journal of information science*, 32(2):198–208, 2006.

[39] Ralph Gross and Alessandro Acquisti. Information revelation and privacy in online social networks. In *Proceedings of the 2005 ACM workshop on Privacy in the electronic society*, pages 71–80. ACM, 2005.

[40] Richard Moe Gustavsen. Condor–an application framework for mobility-based context-aware applications. In *Proceedings of the workshop on concepts and models for ubiquitous computing*, volume 39, 2002.

[41] Lee Humphreys. Mobile social networks and social practice: A case study of dodgeball. *Journal of Computer-Mediated Communication*, 13(1):341–360, 2007.

[42] Jadwiga Indulska and Peter Sutton. Location management in pervasive systems. In *Proceedings of the Australasian information security workshop conference on ACSW frontiers 2003-Volume 21*, pages 143–151. Australian Computer Society, Inc., 2003.

[43] Younghee Jung, Akseli Anttila, and Jan Blom. Designing for the evolution of mobile contacts application. *Proceedings of the 10th international conference on Human computer interaction with mobile devices and services - MobileHCI '08*, page 449, 2008.

[44] Mik Lamming and Mike Flynn. Forget-me-not: Intimate computing in support of human memory. In *Proc. FRIEND21, 1994 Int. Symp. on Next Generation Human Interface*, page 4. Citeseer, 1994.

[45] Cliff Lampe, Nicole Ellison, and Charles Steinfield. A face (book) in the crowd: Social searching vs. social browsing. In *Proceedings of the 2006 20th anniversary conference on Computer supported cooperative work*, pages 167–170. ACM, 2006.

[46] Paul J Leach, Tim Berners-Lee, Jeffrey C Mogul, Larry Masinter, Roy T Fielding, and James Gettys. Hypertext transfer protocol–http/1.1. 1999.

[47] Cameron Marlow, Mor Naaman, Danah Boyd, and Marc Davis. Ht06, tagging paper, taxonomy, flickr, academic article, to read. In *Proceedings of the seventeenth conference on Hypertext and hypermedia*, pages 31–40. ACM, 2006.

[48] Peter Mell and Tim Grance. The nist definition of cloud computing. 2011.

[49] David Millen, Jonathan Feinberg, and Bernard Kerr. Social bookmarking in the enterprise. *Queue*, 3(9):28–35, 2005.

[50] Trung Van Nguyen and Alice Hae Yun Oh. Users' needs for social tagging and sharing on mobile contacts. In *Proceedings of the 12th International Conference on Human Computer Interaction with Mobile Devices and Services*, MobileHCI '10, pages 387–388, New York, NY, USA, 2010. ACM.

[51] Paul Prekop and Mark Burnett. Activities, context and ubiquitous computing. *Computer Communications*, 26(11):1168–1176, 2003.

[52] Nick S Ryan, Jason Pascoe, and David R Morse. Enhanced reality fieldwork: the context-aware archaeological assistant. In *Computer applications in archaeology*. Tempus Reparatum, 1998.

[53] Bill N Schilit and Marvin M Theimer. Disseminating active map information to mobile hosts. *Network, IEEE*, 8(5):22–32, 1994.

[54] Thomas Strang and Claudia Linnhoff-Popien. A context modeling survey. In *Workshop Proceedings*, 2004.

[55] Jenny Sundén. Material virtualities: Approaching online textual embodiment. 2003.

[56] Mark Weiser. The computer for the 21st century. *Scientific american*, 265(3):94–104, 1991.

[57] Steve Whittaker, Quentin Jones, and Loren Terveen. Contact management: identifying contacts to support long-term communication. In *Proceedings of the 2002 ACM conference on Computer supported cooperative work*, pages 216–225. ACM, 2002.