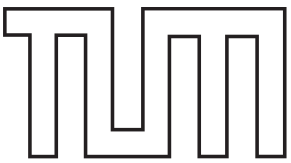

Technische Universität München

Distributed Multimodal Information Processing Group

Prof. Dr. Matthias Kranz

Bachelor Thesis

Location Based Indoor Services

Author:	Lukas Murmann
Matriculation Number:	[redacted]
Address:	[redacted]
Advisor:	Andreas Möller
Begin:	??. Mai
End:	??. September

Abstract

The widespread adoptions of smartphones in recent years lead to a change in how and where people use software applications and services.

As people begin to use a variety of devices in different locations, new opportunities for context aware applications and services emerge. Major web sites and social networks already integrate location-based services in their products.

However, those applications focus on outdoor scenarios. As soon as a user enters a building, they lose precise localization, maps and information about interesting places. There is a lack of well documented, open-source libraries and frameworks that provide the infrastructure needed to develop a location-based service in indoor scenarios.

This work describes a localization solution with room-level accuracy that can be deployed in any building with WiFi infrastructure. Furthermore, solutions for indoor mapping and indoor POIs are provided.
Based on that is Ubiversity, a location-sharing that demonstrates the usage of these generic building blocks. It supports floor plans for mapping, a room database and builds up a reference data set for localization online, with no need for an initial training phase.

Abstract

Die zunehmende Verbreitung von Smartphones verändert die Nutzung von Softwareandwendungen und -Diensten. Softwarenutzung erfolgt zunehmend mobil, wodurch Möglichkeiten für kontextsensitive Dienste und Anwendungen eröffnet werden. Dieser Trend spiegelt sich auch in der zunehmenden Verbreitung kommerzieller location-based Services durch bekannte Internetmarken wieder.

Jedoch nehmen aktuelle location-based Services eine Nutzung außerhalb von Gebäuden an. Innerhalb von Gebäuden gehen wichtige Merkmale wie präzise Ortung, Kartenmaterial und Informationen über nahegelegene interessante Orte (POIs) verloren. Da keine freien, dokumentierten Softwarebibliotheken und Datenquellen für indoor Dienste verfügbar sind, werden Forscher und Softwareentwickler gezwungen diese grundlegenden Bausteine für ihre indoor LBS von Grund auf neu zu entwickeln.

Die vorliegende Arbeit beschreibt eine raumgenaue Lokalisierungslösung. Sie ist in jedem Gebäude mit hinreichender WLAN-Abdeckung nutzbar. Ferner werden Werkzeuge und Bibliotheken für die Erstellung von Raumplänen und POI-Datenbanken vorgesellt.
Aufbauend auf diesen grundlegenden Bausteinen wurde Ubiversity entwickelt. Ubiversity ist ein location-sharing Dienst für Studenten und Forscher an der Technischen Universität München. Die Anwendung bindet Raumpläne ein, bietet eine Raumdatenbank als POI-Quelle und ermöglicht eine WLAN-basierte Ortung ohne vorherige Trainingsphase. Die Referenzdatenbank für den Lokalisierungsalgorithmus wird im laufenden Betrieb aufgebaut.

Contents

Chapter 1

Introduction

Location-based services are one key factor for the success of smartphones and mobile ecosystems in recent years. Mobile device platforms come with tools and libraries that provide common infrastructure for location-based services, thus enabling developers to easily write their own LBSs.
Chapter 2 introduces some of the most successful location-based services. Furthermore, it takes a look back and examines their origins in the research on context-aware computing that was conducted during the nineties at places like Olivetty Research Labs or Xerox PARC.

In chapter 3, we identify the most important components of modern location-based services. First, they have to determine the user's location. This is usually done using GPS or based on nearby cell towers.
Second, this location must be visualized. Web services like Google Maps exist that developers can integrate into their own applications.
But maps are not the only way to visualize information. Geocoders look up interesting places near a coordinate pair and can thus provide a semantic link; users can understand location information more intuitively.

This work focuses on location-based services in indoor scenarios. While libraries for outdoor LBSs are readily available, there is no such support for indoor services.
Indoor localization is still a patchwork. As cell towers are too coarse for many applications and GPS is not available indoors, developers usually have to set up a custom WiFi-based localization solution.
Map services like Google Maps do not have access to indoor maps and floor plans. There are few buildings where indoor maps are publicly available.
Finally, POI databases like Google Places are too coarse and lack the level of detail needed for indoor services. Indoor services usually require POIs like rooms, elevators or coffee makers, rather than the cafés and restaurants usually found in public databases.

Chapter 4 evaluates possible designs for reusable LBS building blocks with special emphasis on the indoor domain.

In chapter 5, those building blocks are developed and then exemplified by Ubiversity, a location-sharing service for Technische Universität München. Ubiversity lets students and staff share their location with friends. It puts special emphasis on state-of-the-art privacy features in order to lower user's reluctance to share their current location.

Chapter 6 documents a field study that evaluates users' reaction to Ubiversity and its privacy approach. The study consisted of two questionnaires, one about the study participant's prior exposure to location-based services and one feedback sheet about Ubiversity and its perceived privacy impact.

Finally in chapter 7, the contributions by this work are summarized. An outlook on possible further research is given.

Chapter 2

Related Work

Before location-based services became successful products in recent years, researchers built the foundation for them under the term *context-aware computing*. In this chapter, we briefly recap the history of research on context-aware computing and some of the most successful location-based applications today.

2.1 Context-Awareness

Research on context-aware applications began in the early nineties, for example at Olivetti Research, England [29] or at Xerox PARC in California [26].
In their 1994 paper on Context-Aware computing [26], Schilit et al. defined the three main aspects of context as "where you are, who you are with, and what resources are nearby". Context is about more than just the user's location. Still, research results on context-awareness computing may be applied on the more special case of location-awareness and location-based services.

It is interesting to note that many early context-aware systems are *indoor* services that use custom localization techniques, while modern location-based service products are meant to be used outdoors on smart phones with GPS localization [29], [26], [14].

2.2 Location-Based Services

Location-Based services can come in many forms: Navigation Software, Restaurant Guides or location-aware information about nearby public transportation. One very successful flavor of LBSs are *location-sharing* services like Google Latitude [13], Facebook Places [27] or foursquare [12].
We now provide a brief overview about those popular location-sharing LBSs and highlight their similarities and differences.

Google Latitude Using Google Latitude [13], users can continuously share their location with friends. Friends will then see the user's most recent location. This behavior differs from checkin based systems like Facebook Places of Foursquare in two ways. First, checkin based systems require explicit user interaction before they publish a location update while Google Latitude can update a user's movements without explicit consent. Second, Google Latitude publishes user locations as lat,lng pairs, so the semantic behind these coordinates is hidden at first. This forms a contrast to checkin based systems, where one can for example check in at *Starbucks*. The precision may be worse than when using coordinates, but the information is much more intuitive to humans using the service.

In recent versions, latitude supports checkin based sharing as well. Users can check in to nearby POIs and configure an auto checkin when Latitude recognizes a place it has checked in before.

Facebook Places Facebook's shot at location based services is Facebook Places [27]. Places' design is tightly coupled to the checkin concept and a major influence for our campus location-sharing service, Ubiversity (see chapter 5.
In Facebook Places, a user sees a list of nearby businesses and other POIs and can than check in himself and friends that are with him. The checkin is only visible to a well-defined subset of the user's friends (using groups) or only to individuals. Checkins always happen at POIs and are displayed in friend's news feeds.

Foursquare Foursquare is in many ways similar to Facebook Places. Users select their location by manual checkin, sharing and social relationships play a crucial part of the service.

What is really unique about foursquare is how it utilizes gamification. Users get awarded badges if they check in frequently at the same place. Once they got badges of many places in a neighborhood, they can earn the rank or mayor and may then get discount at local foursquare-partners.

2.3 Location-Sharing at Technische Universität München

In chapter 5, we describe Ubiversity, the implementation of a location-sharing service for students at Technische Universität München, in chapter 3 and 4 we describe the platform our service is based on.

Chapter 3

Components of a Location-Based Service

Our work describes a reusable platform for location based indoor services. Before we can start designing (Chapter 2) and implementing (Chapter 3) this platform, we first have to identify the basic components common to all location based services.

We obtain this set of components by examining existing (outdoor) LBS platforms as well as specific service implementations. The difficulty here is to provide the features the majority of indoor LBSs can make use of, without getting distracted by specific application logic and features only a handful of services will need.

Mobile device platforms (e.g. Android and iOS [18], [15]) already support developers with libraries for outdoor LBSs. Both make a set of localization techniques (Based of Cell Networks, WiFi or GPS) available to service developers. In addition, both make it easy to embed a map in custom applications in order to visualize location information.

The building blocks we identified provide the basic framework for real-word services. However, custom application logic or infrastructure code specific to a specific application still has to be written. Still, the developed framework is supposed to speed up the development of new application as developers can focus on new contributions, instead of reinventing common infrastructure over and over again.

3.1 Localization

An application that wants to provide location-aware services to the user first has to determine his location. There are different well established techniques, but the cost and possible deployment scenarios vary. An overview on localization-techniques is given in [22].

However, our target scenario further adds constraints like indoor localization, out of the box deployment and clients running on smartphones (see section 3.3). This limits the set of viable localization technologies significantly.

- The Global Positioning System - GPS
- Phone Signal / WiFi localization by third party
- WiFi Fingerprinting with local reference data
- User explicitly picks location from a list or map.
- User scans a NFC Tag with known location
- User Scans a QR Code with known location

What are the advantages and disadvantages of each technique?

3.1.1 GPS

GPS is a satellite based localization system. It is globally available, does neither require users to pay any fees, nor does is require any communication with the service provider. The computation is done *offline*.
Offline computation becomes important if the data connectivity of your devices is limited or for privacy sensitive applications.
However, GPS is not an option for indoor services. It requires line of sight to at least four satellites

3.1.2 Network-based localization

Localization based on a user's WiFi context or nearby cell towers usually happens on a remote server run by a third party. The built-in Android localization stack communicates with Google's WiFi Database, iOS devices send user data to Apple's servers and other apps may rely on a service like Skyhook[19] to convert the incoming WiFi-Signal they receive from access points into absolute coordinates.

Third-party services usually build up their reference database by *Wardriving*, so they only have reference points close to streets. This makes third-party services difficult to use for indoor-localization. Some services though offer the possibility to manually add the location of an access point to their data base (e.g. Skyhook [19]). This might improve indoor localization results with a service like that.

There are commercial products that provide the localization algorithms, but let service developers run their own fingerprint database (Ekahau localization system [16])
Furthermore, active research is continuing in the field of indoor localization, so the study of current publications can serve as a starting point for the development of a custom WiFi localization solution.

3.1.3 User Picks location

Especially for services that share the user's location with other users or may even publish it publicly on the web, the *Check-In paradigm* is in widespread use.
To perform the actual check-in to a Restaurant or Shop however, the last step in the localization process is done by the user. He adjusts and confirms the recommendation given by the system before he finally publishes his location.
See 2.2 for examples of services that make use of this technique.

3.2 Maps

We determined the location and got our pair of latitude and longitude (and maybe even elevation). Still, *"Lukas is now at 48.143346, 11.578259."* isn't really a good way to interact with the user of our service. To interact with a user, it is better to visualize the abstract coordinates on a map. For outdoor applications, there are numerous services and map providers we can choose from. The options range from integrating a web service like Google Maps to working with the raw map data from commercial (e.g. Navteq [24]) or free map sources (OpenStreetMap [11]).

However, there are few solutions for indoor mapping and even fewer are available for free. If developers want to use indoor maps in their applications, they have to include a floor plan of their test environment in order to visualize coordinates on a map. One challenge with this is that developers usually do not know the exact location and rotation of the floor plans they have available what makes the projection of coordinate pairs on the floor plan quite cumbersome.

3.3 Client Hardware

The localization techniques we described in section 3.1 require a dedicated client device the user can interact with. It is in the very nature of location-based services, that they must be used *mobile*, not on general purpose desktop machines.

Early location-aware systems were even named after those client devices. In 2 we mentioned the Active Badge system [29], where users had to wear a badge on their shirt that would then be recognized by installed infra-red scanners. The context-aware systems at Xerox PARC made use of the Pad and Tab devices that were created there as part of the ubiquitous computing vision [30]. Back then, researchers had to design custom client hardware for their services.

The widespread adoption of smartphones changed that. Platform providers provide libraries and documentation, enabling researchers and developers to write custom applications for their devices.
In the next chapter we'll evaluate available mobile device platform and choose one for our example location-sharing service.

3.4 Points of Interest

Maps are not the only way to communicate coordinates to a user. We can translate the abstract coordinates to concrete Points of Interest like an address, the name of a shop or a room number. This translation is called *geocoding* and relies on a database with Points of Interest suitable for our service.

Again, for outdoor services, there are POI sources and geocoder already integrated into mobile platforms (Google GeoCoder, iOS NSLocation Framework). But do third parties know relevant indoor POIs like *"The cafeteria in building B"*? Probably not, so geocoding and POIs are another piece of infrastructure that needs to be provided when developing indoor LBSs.

3.5 Communication

Most Location Based Services require a communication channel with a server or other clients. For instance (see 3.1), fingerprinting based localization algorithms rely on a fingerprint database on a server.
Static data sets like map tiles or POIs can be stored locally on the device, but there may be cases when it is desirable to update those data sets with information from the server as well.
In many cases, Location Based Services don't just consume information. They are a source of information as well. In order to communicate this information to the server or other users, the service must establish a communication channel.

Our choice of communication channel is based on two factors. First, the capabilities of our devices (smartphones) and second, the nature of the messages we plan to send. For example, a wireless sensor node that periodically sends measurements requires a different communication protocol than media streaming to a multimedia device.

The data that clients in a location-based service send to the server can in general be represented using text representations. For example, information about a point of interest can be represented as numbers (for coordinates) and text that names and describes the POI.

The communication channels used most widely nowadays are web services. For our purposes, the definition of web service is
A service that sends and receives information

- using the HTTP protocol [9]
- using text representations of data as the primary message format

One popular subclass of web services are so called RESTful web services. REST is an architectural style introduced by Roy T. Fielding in his 2000 doctoral thesis [10].

3.5.1 Why a RESTful communication

Connection to other services There are many other services that provide RESTful APIs that can serve as information sources or sinks. Especially for tasks like geocoding there are many services that provide RESTful APIs free of charge.

Device support All mobile devices ship with libraries that support sending HTTP requests. This is all you need to start developing against a RESTful API. Choosing REST as our communication style, we can deploy our services to a variety of devices and do not lock us into a vendor specific technology.

Web Frameworks Another reason for the popularity of RESTful services is the good support for server development. Web frameworks like *Ruby on Rails* or *Django* emerged that provide solutions for many common tasks such as user management, logging or data abstraction.

3.5.2 Downsides of choosing REST

The design decision to choose REST as our communication style comes at a cost. HTTP carries some overhead to establish a connection to the server. It is layered on top a reliable TCP connection what might increase the response time of your network communication. Examples when HTTP might be the wrong communication protocol for your application include

- Clients that send very frequent status updates where a single status update is not important
- When network traffic is very expensive

In those cases, a custom communication protocol without the overhead of generic protocol headers or reliable connections could be a better choice.

Chapter 4

Designing the LBS Components

After identifying the most important components a location-based indoor service relies on, we now continue with a more detailed description of each component's design. We take a look at competing technologies, evaluate their specific strengths and weaknesses and choose the ones that allow for easy and fast development of location-based development projects.

4.1 Localization

As described in section 3.1, localization is an important component our platform for location based indoor services must deliver. We already discussed the most common localization techniques:

- GPS
- Localization by third-party service
- WiFi-based localization with custom reference dataset
- Explicit localization by the user - checkin

Our focus on indoor services rules out GPS as a viable localization solution. It could at most serve as a very coarse recommendation (e.g. preselect a building) for explicit user localization, but as our platform aims to provide localization with room-level accuracy, we need to look out for a more precise technique.

WiFi localization using a third-party service like Skyhook [19] may work in some environments where nearby access points have already been submitted manually and this data is actually used by the service.
However, developers have little control if and when their submitted data will actually

show up on the live service so for our general purpose platform, we want to provide a more reliable solution.

4.1.1 The platform's localization concept

WiFi Fingerprinting The most important part of our localization solution is a WiFi fingerprinting algorithm. Users can scan their current WiFi context with their device and send this data to the server, which will then respond with a list of nearby POIs.
The necessary training data for this fingerprinting technique can be submitted using web-requests until there is a good coverage of the area the indoor service will be deployed to.
It is important to note, that the localization component relies on a list of POIs. As our goal is to provide room-level accuracy, we do not map fingerprints to x-y-z Coordinates, but map them directly to room numbers (POIs).

Manual Checkin In addition to fingerprinting, the localization component supports manual localization by various means. For example, the user can pick a room from the list of POIs. This list may be filtered by building and floor based on previous WiFi Localization.

NFC Tags We also support Localization by *touching* NFC tags. These tags may be installed on door signs and when touched with the phone, send the identifier for a POI to the client software running on the device. This identifier can then be used to look up additional information from the POI database.

QR Codes For devices that do not support NFC, we also support POI identification based on QR Codes. Again, the QR code contains a unique POI identifier that is sent to the client application once the user has scanned a code with the device camera.

4.1.2 Localization Wrap-Up

By combining Fingerprinting-based localization with explicit user localization, our platform gains an important characteristic: There is no need for an initial training phase that is needed for plain Fingerprinting-based deployments..
Whenever the user checks in using manual selection, the client phone scans its WiFi context and sends a new (Fingerprint, POI) tuple to the server that can subsequently be used for localization.
This *on-line training* renders the initial training phase unnecessary and is one feature of our platform that allows for fast deployment on new sites.

4.2 Map

In section 3.2, we already pointed out the importance of maps to location-based services. A localization algorithm usually calculates absolute coordinates that are great for computation, but unintuitive when displayed to an end user.

Our LBS platform supports two means to attach meaning to coordinates. First, we show how our platform helps visualizing them on an indoor map and in section 4.3 we explain how the platform handles POIs that attach semantic meaning to coordinates.

For outdoor LBSs, mapping is already provided by mobile device platforms or by a web service such as Google Maps. Indoors however, those services are of little help. Their precision is too coarse, they cannot deal with multi-story buildings and the floor plans of most indoor facilities are unknown to them.

Our goal was to provide a map component that is as easy to use as third-party map services, but lets users of the component import their own maps and floor plans. The client software will contain a module that combines (*stitches*) multiple floor plans into a consecutive piece, always layering the most detailed map on top of coarser ones.
The map component will be able to display POIs or user locations at coordinates provided as latitude / longitude pairs, just as known from the mapping libraries that are built into most mobile device platforms.

However, in order to display information on floor plans correctly, one must first specify their location and scale. The map component can then use this information to project (lat/lng) pairs to the correct pixel of the floor plan.
Usually, one does not know the exact scale and orientation of such a plan. In order to correctly display our POIs, we must find the transformation matrix that maps the real world's (lat,lng) space to the (x,y) plane of the floor plan.

We now look at a technique that calculates this transformation from three reference points. A tool to calculate and export the transformation matrix for each floor plan comes with our LBS platform.

A transformation in 2D space can be expressed as a 3x3 Matrix using homogeneous coordinates. Our Problem can be formulated as

$$Ax = b \tag{4.1}$$

with x being the $(lat, lng, 1)$ coordinates and the result $b = (x, y, 1)$ representing our floor plan coordinates. In order to solve this equation we need three linear independent (lat,lng) and (x,y) pairs, so the developer must know the exact coordinates for three points on each of floor plan.

The software tool that comes with the platform lets developers load floor plans as image files and specify three known reference points in a point-and-click fashion. The software then calculates the 3x3 transformation matrix A which lets the client software map the geo coordinates received from the localization module to floor plan coordinates.

Using this transformation matrix A, the client can then display the user's location in a more visual and intuitive manner.

4.3 POI Dataset

We described how we can visualize coordinates by using custom floor plans. Often however, coordinates don't just represent an arbitrary point on a map, but a specific place that *is of interest*.
Once we determine one or more such points of interest that correspond to (lat, lng) pairs, we can display a location in a much more concise way than on a map. For example, we could build a service that looks up the user's current location and then lists information about nearby POIs like rooms, printers or coffee makers.

POIs can often be structured in categories. Those categories help to display only the subset of information to the user that's relevant in the current context. Our location-sharing service for example filters rooms by campus, building and floor, but other services may use custom categories that fit their problem domain and deployment scenario. The chapter on the implementation of the room database (5.7) provides information on how to provide POIs databases both on the server and locally on a device.

The platform can store POI information both locally on the device and on the server. If the set of POIs does not change over time, storing the database locally on the device safes network bandwidth and increases the responsiveness of the client application. The location-sharing service for example stores general POI information on the device, so information about location, room name or room capacity is instantly available without the need for a network connection.
We can then supplement this general information with updated information from a web service. For example when a user scans the door tag in front of an office room, the software could look up the office owner's public calendar or phone number.

4.4 Communication - A RESTful approach

We identified the need for a communication channel in section 3.5. Furthermore, we evaluated popular communication channels on current smart phone platforms and settled on a RESTful communication approach.
The most crucial building blocks of a RESTful web service are

- Resources
- Representations

We give a short explanation of these central terms in the next sections. See [10] and [25] for a detailed discussion of the REST architectural style.

4.4.1 Resources

A resource is something that is of interest to the clients of our web service. This definition is very vague, so to give an idea what resources could be, here are some of the resources that are exposed by our location-sharing service:

- Rooms
- Buildings
- Users
- A User's most recent location.
- A User's friend list.

The resources other services need to model may differ from the example but for most applications, POIs and POI categories are definitely a good starting point to begin model RESTful resources. Once we figured out what resources we want to make accessible, the next step is to define *representations* for those resources.

4.4.2 Representations

A resource is a rather abstract concept. We still don't know how resources are represented on disk or how they are serialized when transferred over the wire. We must define concrete representations that help us retrieve and update our resources.

A representation may be anything from plain human readable text to binary data (for example for image resources). As our resources must be interpreted by the client, we'll want to serve them in a machine readable format. When RESTful web services first emerged, most of them used XML [4] as their representation format. XML parsers are available on all mobile platforms, well supported by web frameworks and client programmers usually know how to decode information that is provided as an XML representation.
More recently, an alternative to XML emerged. The Json format [7] has a simpler structure than XML and is very easy to create and parse on most platforms. Many major web services now offer Json representations of their resources in addition to XML representations. Some even completely switched their services to Json representations.

For our reference service, we chose Json as our primary representation format.

4.5 Client Platforms

We just (4.4) defined a communication channel and the format of the messages we exchange over this channel. Next, we must settle on a specific technology for our clients. In (3.3), we identified the need for client devices and further identified smartphones as a flexible and popular platform to build on.
This section will give a short overview on available mobile device platforms and describe the reasons that let to our decision for one of the platforms.

Since Apple launched the iPhone in 2007, many mobile device platforms compete for customers and, as customers began to expect customized native applications, also compete for developers that support their platform.
Researchers in the field of location-aware services benefit from this competition. There is a variety of platforms and devices available. Platform providers want to make programming applications for their platform as easy as possible, so developer tools and API are very well documented [15][18].
A non-exhaustive list of competing platforms at the time of this writing:

- Apple iOS
- Android
- Windows Phone
- BlackBerry

As those platforms all provide different libraries and programming languages, developers have to make a decision on which one to support. All platforms have similar technical capabilities and all provide good developer support and documentation, the choice is more

a matter of personal taste, availability of test devices or policies of the developer's organization.

For our reference implementation, we settled on Android [18]. Android's open-source nature makes it an interesting platform for academic use as the implementation of the platform itself can be studied. Furthermore, it is easy to deploy the software on test devices or even publish it on the android market.

4.6 Server

We defined a communication channel, we settled on a representation format our clients will understand, the one thing missing now is the server they communicate with. Luckily, there is a variety of frameworks and middleware for web services to choose from.

Those web frameworks help web developers with common tasks such as account management, serialization and encoding of network messages or database abstractions.

Examples for web frameworks include

- Rails
- Django
- Microsoft ASP.net

Again, the choice of web framework is more an implementation detail and may again depend on personal taste, prior experience using a related technology or organizational policy. The web framework we chose for our platform is Django. Django is an open-source project written in Python that comes with a well-written online documentation [1]. Django combines several strengths that make it a good choice for our LBS platform:

Open Source As an open source project, one can learn about the implementation of the framework by browsing the source code, or even contribute back improvements or additions to the Django framework.

Object-Relational mapping Django comes with an object-relational mapper. We can write special *model-classes* that can then be persisted to a database. Django supports a variety of relational databases and there are efforts to support non-relational DBMS [3] as well.

Easy to get started Django puts emphasis on the framework approach. It comes with a tool that creates the project structure and configuration files developers need. This approach may limit Django's flexibility for very specialized tasks, but it helps to get up to speed quickly and to focus on new contributions instead of common infrastructure. As this is the same approach we pursue for our platform in general, Django's design philosophy integrates well into our LBS framework.

Privacy

Location-based services handle very sensitive personal information.
By default, HTTP requests are sent unencrypted. Recently, it has been demonstrated [5], that when using an open WiFi-Network, personal information can be intercepted easily without specialized equipment or technical knowledge. Consequently, all traffic sent between clients and server of our application is encrypted using SSL/TLS [8].

But privacy is not only about eavesdropping or man-in-the-middle attacks. We want to give users control about the information they share with other users. Therefore, we implemented a custom user-account system and permission-middleware so users have fine-grained control about the information they share. See 5.5 for more details on how we use a versatile permission system to model complex dynamic relationships between users.

Chapter 5

Ubiversity - Example Location-Sharing Service

In Chapter 1, we identified the components almost all location based services rely on. In the second chapter, we went into more detail, took indoor location as a constraint into account, specified the features we want to implement and settled on specific technologies that best fit out needs.

In this Chapter, we describe the implementation of each component and an Ubiversity, an example service that uses the identified building blocks. The Ubiversity app contains both social and location-based features. It can serve either as reference for completely new LBS, or as a starting point for customization.

Ubiversity is a location sharing service for students and staff at Technische Universität München. It lets users *check in* in order to share their current location with their friends. Friends can then see the checkin using one of the supported visualization techniques: The check in is visualized as a marker on the floor plan and shown in a feed, together with meta information on the room where the friend checked in.

Apart from providing an interesting example service for the platform described in the previous chapters, the goal was also to gather information on how students would use such a service. We conducted a field study that gathered information on how people used our service. The main question the study tried to answer was if they felt comfortable sharing personal information when using this location-based service. The study's design and results are discussed in chapter 6.

5.1 Motivation for Building a Location-Sharing Service

We chose location-sharing because it requires both context-awareness and an elaborate account and permission management. The permission system (see 5.5) is used to give users fine-grained control about the circle of users they share each individual check in with. Location-Sharing also enables us to connect the client-application to its environment, allowing location-information to be gathered from various sources like WiFi Fingerprinting, NFC tags or QR Codes.

In chapter 2, we introduced three of the most successful location-sharing services. Before we go into more detail on noteworthy aspects of the Ubiversity application, we first highlight the similarities and differences of the existing services to our location-sharing app.

5.2 Comparison to Related Location-Sharing Services

Even though major internet brands are offering location-based services (see 2.2), people are often reluctant to use them. This is at least the author's impression from interviews with students that are otherwise tech-savvy and regular users of social networks (see 6.3). Information about a person's current location is very personal and people are not comfortable sharing this information on a commercial and partly public platform.

This observation was the motivation to design a localization service with limited scope. The servers of our example service are hosted at the Distributed Information Processing Group; people using the service are students and staff: This creates a much more local and personal user experience.

5.3 Communication - Resource Design

As described in section 5.8, we want to implement a RESTful API to our web service. REST is all about resources and methods (HTTP GET, PUT, POST and DELETE) operating on them. A good way to design a RESTful API is to think about the resources a service deals with.

Rooms The most common POIs in our checkin service are rooms, so we expose a `/room:<id>` ULR with the id being an integer following a university-internal numbering scheme.
We further expose `/rooms?<mac_addr>=<RSS>&<mac_addr>=<RSS>`. The query string provides the input for our WiFi-Based localization algorithm; the response will contain a list of rooms and a similarity quantifier.

Buildings What other Resources do exist? There's `/building:<id>` that exposes more detailed information on the building of a room like number of floors or the building's zip code.

We introduced the POI-related resources, what other resources does a checkin service need?

Users As mentioned in the introduction of this chapter, we put emphasis on account and permission management to facilitate robust and fine-grained privacy features. All of those features center around `/user:<id>` resources or `/users?query=string` resources that lets you search for users based on their name and email. The `/users` resource also allows user creation by sending a POST request with necessary account information.
`/user:<id>` resources require log-in and do only yield information once the user sending the request was added to the requested user's friend list. Users can request their own profile information by sending an authenticated request to the special `/me` resource.

Friends In our example service, a user's friend list is only visible to the user himself. It is located at `/user:<id>/friends` and responds both to GET (returns list of confirmed as well as requested friendships) and POST. POST requests a friendship or approves a pending friend request.

Feed All friend checkins a user can access are aggregated in the feed. I can be retrieved from `/user:<id>/friends/feed` and is a chronological list of the latest checkins that were shared with the user.

This section provided an overview about the most crucial resources of our REST API. There are more that provide shortcuts or more detailed information.

5.4 Database Schema Design

We make use of the object-relational mapper (ORM) that is integrated into the Django web framework (see 4.6). As data store, a MySQL database is used. Using Django speeds up development as no custom SQL code has to be written. All developers have to do is to write their model classes in Python. The necessary SQL commands to access the database are then automatically generated by the Django framework.

Our models roughly correspond to the resources we modeled with our RESTful API. There are for example classes for rooms, buildings, users, or groups. Objects of these special model classes can be persisted to the database.
The general rule of thumb here is that each class corresponds to one table in our database; each object to one row. We can define relationships (1:1, 1:n, n:n) in our model classes that are correctly mapped to the database.

The permission system is the most complex part of our model layer. Permissions can be attached to both groups as well as individual users. We derive the User and Group classes from the same base class and assign permissions to the base class instead of Users and Groups directly (see figure 5.5). Luckily, the Django ORM supports this kind of inheritance. It creates all the necessary tables for base and sub classes.

This has provided a general overview about what is possible with the Django ORM used by our LBS platform.
The next section provides a more detailed look on permissions and how access to a resource is granted once a user was given all necessary permissions.

5.5 Permission System

The requirements of our permission system exceed the capabilities of the permission system that ships with Django. The Django system is limited as it only supports *per model-class* permissions. For example, a user may only be granted a *view-rooms* permission. This will allow him to view all rooms; by default it's not possible to give object-level permissions.

However, for a privacy-centeric checkin system, we need a much more fine-grained level of control about permissions. By default, a checkin is visible to all of a user's friends. However, a user may limit this visibility to a number of groups, or even a number of individuals from his friend list.

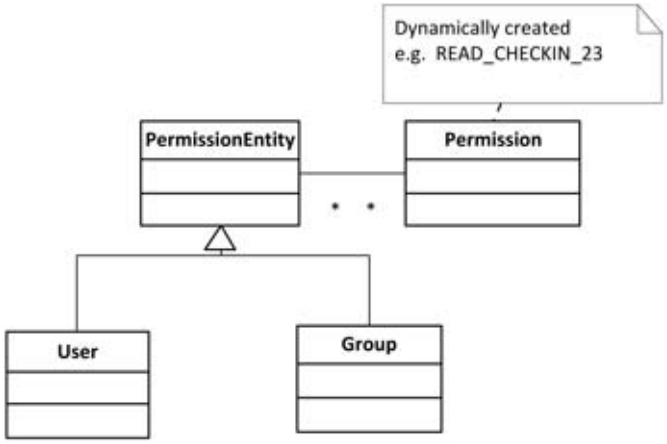

Figure 5.1: The User and Groups class both inherit from the PermissionEntity super class. This class is modeled in its own SQL table.

We cannot just grant *permission to read all checkins of user A* to A's friends, but rather have to create a new permission on each checkin and then grant the *"permission to read checkin X of user A"* to all friends or to a subset of them. This way, not all checkins must be visible to the complete friend list. Figure 5.5 visualizes this permission inheritance.

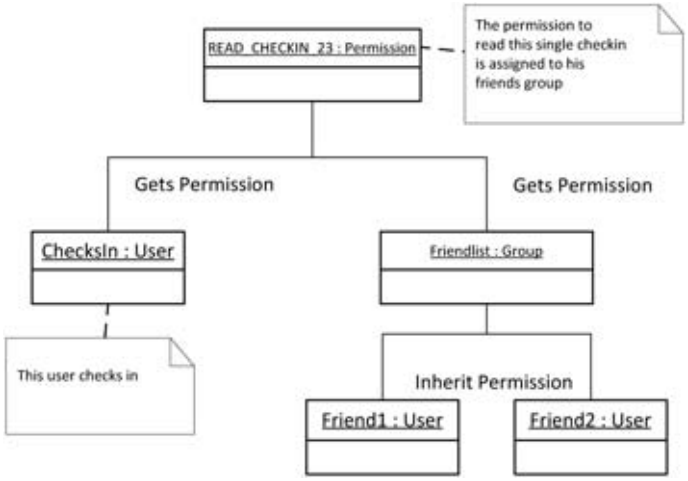

Figure 5.2: When checking in, the Friendlist group is granted the permission to view this checkin. Members of the friend group inherit the right to view the checkin for as long as they are members of the group.

Checkins are only one example that requires such an elaborate system. Other applications using our platform can reuse the permission system and adapt it to their specific needs.

5.6 Creating Floor Plans

After discussing API design and how to grant users access to resources on the server, we now focus on visualizing location information. In 3.2, we identified maps as a common way for LBSs to present such information. Later (4.1.2) we introduced an algorithm that provides us with the scale and orientation of our existing floor plans.

The platform comes with a tool that calculates a 3x3 transformation matrix from 3 reference points we specify on a floor plan. The workflow how to create those matrices and how to export them to a XML format can be summarized as:

1. Load a map (.jpg, .png or .bmp)
2. Drag all three pins to a recognizable location on the floor plan (see figure 5.6).
3. Enter latitude and longitude for each pin.
4. Click save. This will save the image file and the transformation matrix to a separate directory.
5. Repeat 1-4 until finished. We can then press *export* to export all transformation matrices to a custom xml format that can be read by the Android client.

This tool has been written in C++ [28] using the Qt [6] [20] framework. It can be used for any project where one has to determine the transformation data of single map tiles. The export functionality exports to a custom XML format that is understood by our Android mapping implementation.

Once we are done calculating the transformation information for each floor plan, we are finished with mapping for now. When we integrate the map view in Android at a later point, we will have to retrieve the image file and transformation xml from the export directory and copy it to our Android resources.
After those preparations for mapping, let's not take a look on how to prepare POI data, the other way to visual location information, in a way that will be recognized by Android.

5.7 Creating the POI Database

As described in section 4.3, POI information may be stored either locally on the device or fetched from a web service. Ubiversity tries to store the most frequently used information about POIs on the device. This is required, as we want our users to be able to quickly skim

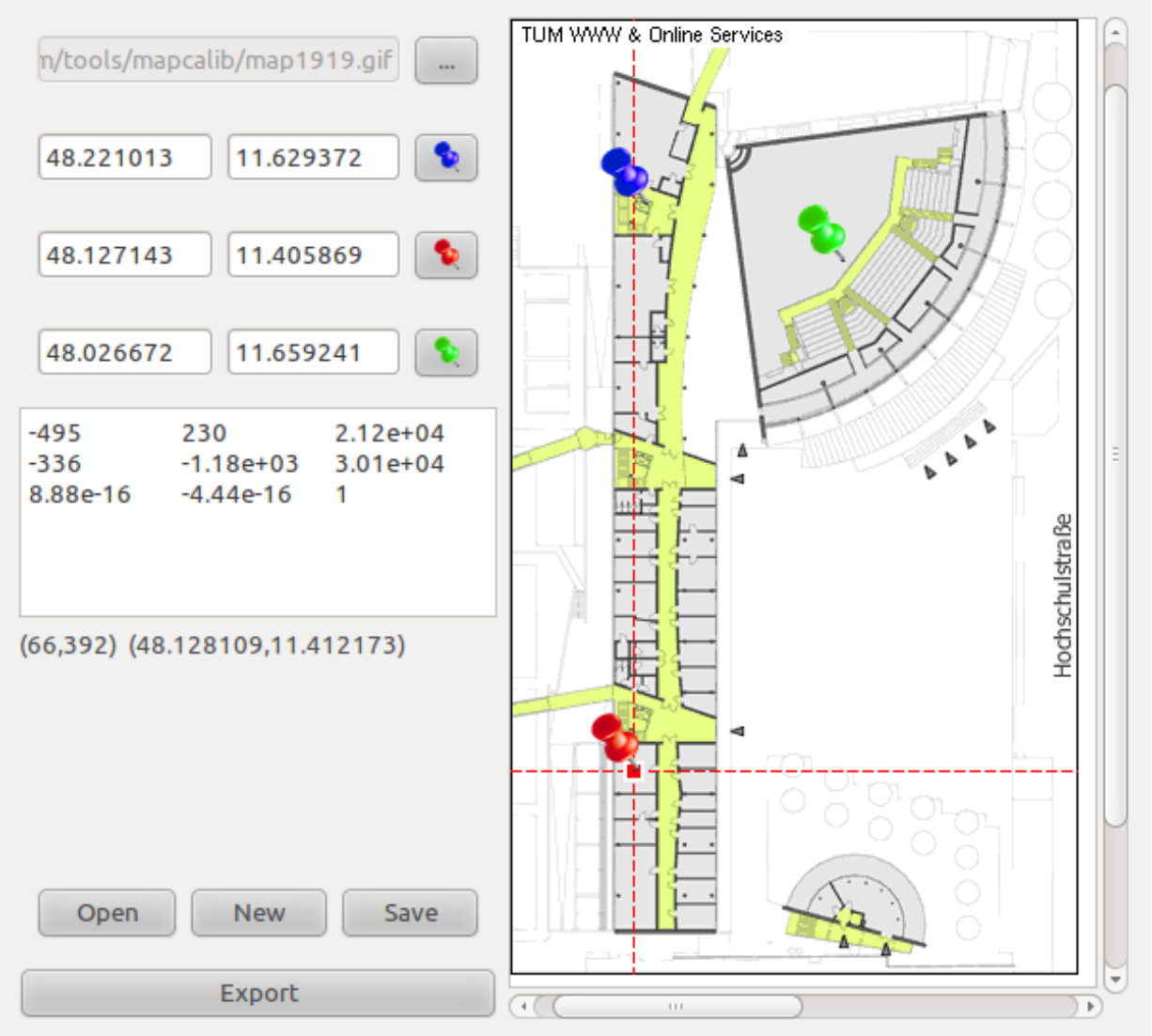

Figure 5.3: Once the user positioned three pins, the calibration tool calculates the transformation matrix of the floor plan.

through a large dataset of rooms (15,000+). Network roundtrips would seriously impact the responsiveness and usability of our client application.

The easiest way to make such data sets available on Android is through a SQLite [2][21] database. SQLite is a lightweight in-process relational database. All tables of one SQLite database are stored in a single file. We can prepare this POI file on a desktop PC and provide it as a resource of our Android project. Figure 5.7 shows an example of the data that is stored in such a database
Our platform comes with a small script that helps creating such a SQLite database.

```
sqlite> select * from rooms limit 5 offset 30;
_id   room  latitude  longitud  name   floor  building              campus
----  ----  --------  --------  -----  -----  --------------------  ------------
7059  307   48.15094  11.56805  N4307  04     N3                    Stammgelände
7061  576   48.14984  11.56669  Z0576  EG     Maschinenwesen (alt)  Stammgelände
7063  308   48.15091  11.56803  N4308  04     N3                    Stammgelände
7067  578   48.14984  11.56669  Z0578  EG     Maschinenwesen (alt)  Stammgelände
7072  580   48.14984  11.56669  Z0580  EG     Maschinenwesen (alt)  Stammgelände
```

Figure 5.4: 5 out of more than 15.000 rows. The most important values are stored on the device. More information can be loaded from the remote server.

Create from Django Models If we already have the POIs modeled in Django, we can quickly choose the tables and columns we want to have available on our devices. We created a script that already handles the creation of a database file and stores our room example in the file. When used for another LBS project, the only thing needed to do is to change the POI model classes in the script to the ones that should be read from the MySQL database.

In this section, we described how to export POIs from Django to a SQLite database file. Later, we will also describe how we access this file from the Android client and make its content available throughout our application.
First however, we have a brief look on the architecture of the Android device platform and examine the most crucial concepts every Android developer must be familiar with.

5.8 Android Overview

So far, we designed the resources of our RESTful API, then had a look at some implementation issues of our service backend and finally prepared maps and POI databases the will help visualizing location data on our client.

Now it is time to get started with our client implementation. This section gives a brief overview on Android application components and highlights the parts of our localization-service that are most interesting for as input for further research on location-based services.

Activity One of the basic building blocks for Android applications are *activities*. Each activity is responsible for a different task an Android application can perform. As users navigate through Android applications, new activities are started and brought to the foreground of the application. Meanwhile, others are paused and sent to the background until they are needed again.

An activity by itself is not visible to the user. Activities merely perform computations, react to user interaction and connect to other components. However, activities control views that can display information to the user.

View Views are the visible components of an Android app. Views know how to render text or images on screen; it is the duty of activities to create or retrieve the content its views are going to present.
Most of the time, Android developers are writing activities. The Android SDK already ships with a huge number of views developers can use, so writing custom views is only necessary if developers need precise control about the looks of an application. Our LBS platform contains one custom view: our map view that transforms and renders floor plans using previously (5.6) determined transformation matrices.

Content Providers Content providers are a much more specialized component than views or activities. They define a uniform interface to access arbitrary data sources from different parts of an application (or even completely separate applications).

A content provider may provide data that was fetched from a web service or access data already managed by the Android system, for example contact data. The most common data source however are SQLite databases. Our example Android client contains a room content provider that is used to access the SQLite database we created in section 5.7.

5.9 Important Activities

After looking at the basic components of each Android application, we now examine some of Ubiversity's activities. While some of the activities are strongly related to the location-sharing service domain, others like the map view or friend list can be of use in many location-based service projects.

5.9.1 Friend Feed

The friend feed is the starting point of our example service. It lists the most recent checkins of a user's friends in chronological order with the most recent checkin on top. The name of the POI that is associated with this checkin is retrieved from the room content provider.

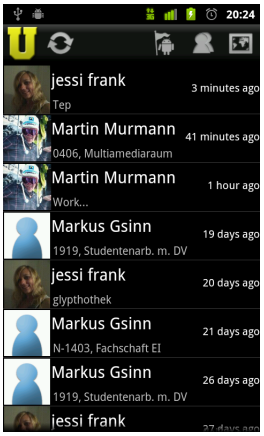

Figure 5.5: The Friend Feed is Ubiversity's central view. It shows checkins of friends in chronological order.

By clicking one of the buttons on the action bar on top, the user can navigate to the three other main activity of our example service: friend list, map view and checkin activity.

5.9.2 Friend List

The Friend List displays all friends of a user, all friendships he requested, as well as all friends request to him. To accept a pending friend request, he can accept it with one click. A click on the cancel button will ask for verification and reject the friend request.

After a friendship has been accepted, users can categorize their friends in groups. Those groups are how users can utilize the complex permission system we implemented on the server (5.5). For our example service, groups are used to make checkins visible to only a subset of a user's friends. But there are many more use cases for user groups and the permission system. Other services could use groups to limit visibility of profile information, control access to admin controls, or even provide premium content based on a user's account state.

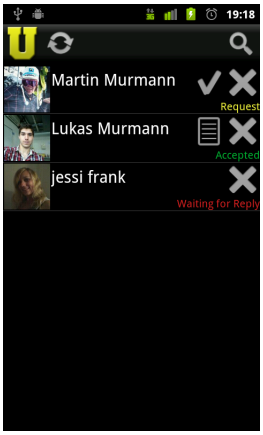

Figure 5.6: The Friend List shows requests and confirmed friendships.

The action bar contains a magnifier-icon that opens the *friend finder*. The friend finder queries the `/users?query=string` resource described in 5.3. The results of this query are displayed in a list and users can then send out friend requests.

This described most of the functionality that's available in the friend list activity. There are two more left: The map activity and the check activity where users can select the POI to check in at.

5.9.3 Map Activity

In the previous chapter (4.1.2), we explained how we can obtain transformation matrices that map the pixels of a floor plan image to the latitude-longitude coordinate space. These matrices are included as resources of our Android application together with the floor plan images.

The floor plans we got however may be overlapping, have different scales or display different levels of a building. We have to choose a subset of the floor plans that can be combined to a suitable map.

Before we can display any floor plans or POIs on the map, we first have to select the set of floor plans we need to display. Therefore we order them by scale (retrieved from the transformation matrix) and see if each floor plan actually contains one of the POIs we want to display.
There is a special class in our Android client that applies a map-selection algorithm and automatically selects a suitable set of floor plans.

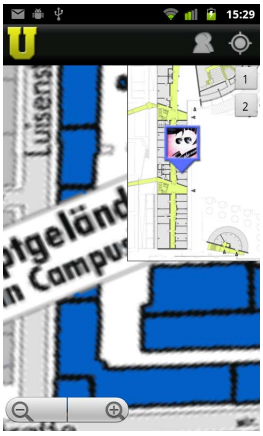

Figure 5.7: The Map View automatically scales and rotates floor plan tiles to create a single map.

Customized icons can be displayed on the map view as overlay. Two kinds of overlays are supported: Interface elements, where users of the class specify the location in display coordinates as well as geo overlays. Geo overlays allow developers using map view in their applications to display an icon on a specific coordinate expressed as latitude/longitude. Ubiversity uses geo overlays to display a user's portrait picture at the position he last checked in.

5.9.4 Check In

We previously (4.3) described how we structured our POIs in categories, namely *campus*, *building* and *floor*. These categories help users to browse through the list of available rooms. They apply three additive filters (one for each category) and are finally left with only a handful of rooms to choose from. As the room database is stored locally, applying filters and scrolling through the results can happen fast and responsive. Figure 5.9.4 shows an example selection for the checkin view.

5.10 Localization

5.10.1 Manual Checkin

As described in section 5.9.4, the simplest way of localization Ubiversity supports is to explicitly select a nearby point of interest from a list. Ubiversity's main POI data set are rooms of several TUM campuses, but the user can check in to places outside of TUM as well. We integrated POIs provided by Google Places ([17]), so users can easily share their

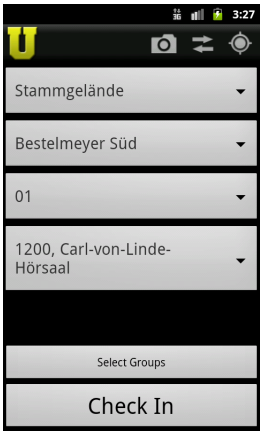

Figure 5.8: In the checkin view, users can either select a POI from a list or enter their current location in a free-text field.

location during their spare time, when they are not at TUM.
Should users be at a place that is neither in our room database, nor listed by Google Places, they can enter their location in a free-text field as well. They can for example use this to share more coarse locations like *at home* or locations that do not map to exact coordinates like *in the bus*.

5.10.2 WiFi Recommendation

As described in section 4.1, manual room selection is not the only way for users to select their current room. In the action bar on top, there is a *localize* button. Once clicked, it records a scan of the device's WiFi context (Received Signal Strength from each access point) and sends the recorded RSS,MAC-address tuples to the `/rooms?mac1=rss1&mac2=rss2...` resource on the server. The response then contains a list of rooms that have an RSS profile similar to the one sent to the server. The similarity between input fingerprint and recorded fingerprint is expressed in a similarity metric. Client application may ask for confirmation by the user should this similarity metric be small or very similar for different rooms.

Location estimation based on WiFi data is a useful tool, as it does not require additional hardware installations, as long as WiFi infrastructure is already in place. However, this technique does not give us absolute certainty about the user location, but rather a good estimate that further depends on the quality and quantity of the WiFi reference data available.

5.10.3 QR Code

In the action bar (camera icon), there is another technique users can choose to select a room: QR codes. For this it is required to place a QR code somewhere inside or before each room (for example the door sign). The QR code contains a unique identifier for that room; in our case there already exists a numbering scheme and this number (integer) also serves as the primary key for the POI database we compiled in 5.7.

The user can then scan this QR code with the device's camera, the unique identifier is passed to the device and the scanned room is selected for checkin.

5.10.4 NFC

Very similar to QR codes, the client application also supports POI identification by scanning NFC tags. The NFC tags must also contain the unique POI identifier, so the difference between NFC-based and QR-based check in is just the difference between scanning a code with the device's camera and holding the device close to a NFC tag attached to a POI.

5.11 Room Database

The creation of the room database was already briefly described in 5.7, we can now load the SQLite file we created earlier into our Android project and look up information about our POIs.
In order to hide the actual data source (the SQLite file), and to allow applications to share data, we implemented a POI content provider (see 5.8) that manages access to the database.

This concludes our chapter on the implementation of the Ubiversity demo application. The next chapter focuses on the user feedback for the app and the concept of a *local web service* for location sharing on campus.

Chapter 6

Field Study

We designed the proposed indoor LBS platform to be independent of third-party services. This enables us to host all required components (localization, mapping, POIs, and user-accounts) on a local server. One of the reasons for this design decision was to deploy services with local scope and limited scale. Our goal is to analyze if users are less reluctant to share their location information in this local setup, than on a global platform.

6.1 Study Design

We conducted a field study with Ubiversity users. During the course of the study, users did

1. fill out a pre-survey about prior experience using location-sharing services.
2. download the application and use it for several days
3. fill out a post-survey about their usage of the service and about any privacy concerns that may have been raised.

In the next sections, we describe the pre and post surveys in more detail.

6.1.1 Pre Survey

In the pre survey, we asked the study participants if they use other location-sharing services. The services we asked about were *Facebook Places*[27], *Google Latitude*[13] and *Foursquare*. We asked two questions on each service. The First question was how often they use the service (answers could range from *never* to *daily*). The second question was how many people can usually see their location when they share on the respective platform.

6.1.2 Post Survey

The Pre-Survey was mostly about our study participant's prior experience with related location-sharing services (2.2). In the post-survey, we then asked how they experienced our local location-sharing service and if the experience differed from prior ones.

We first asked general questions about age and gender, profession and field of study. Participants were allowed to skip these questions, as it may be possible to identify individuals from this data.

General Question Design Most of our questions asked for the participant's opinion or feelings. Other questions asked for data that user's would most likely memorize correctly. Consequently, the answers to the studies questions used a Likert [23] scale. Questions were formulated as statements like "I checked in frequently using the app", users could then answer on a scale ranging from 1 (completely disagree) to 5 (completely agree).

This concludes they general design of our study and the design of both surveys. The next section describes how we acquired our participants, deployed the Android application and enabled participants to anonymously submit answers to pre and post survey.

6.2 Study execution

The main use case of our service is *sharing*. This implies that the participants need at least one friend or colleague using our service as well, so they can share their location with him or her. Furthermore, our goal was to evaluate how the usage behavior of our participants developed over a time span of several days. Would the service slowly *grow on them*, as more and more of their peers begin sharing their location? Or would they use this shiny new thing heavily in the beginning, but then lose interest after some days?

These kinds of questions cannot satisfyingly be answered in a controlled laboratory environment. Therefore we opted for an open field study as the overall model of our study. Students and research assistants interested in the study were given the address of a website documenting the study. There, we presented them a short manual to Ubiversity, as well as the general outline of the study. Participants were asked to submit the pre-survey online, before they continued with downloading and using the application.
After some days of usage, they were again asked to return to the website in order to submit the final post-survey.

6.3 Study evaluation

We made sure participants could anonymously submit their answers. In order to collect more specific and authentic opinions, we also conducted an interview with three of the participants. These interviews were an important addition to the generic survey answers, as they allowed participants to share feedback *off the record* and go into more detail about possible design, usability, or privacy issues.

This section focuses mainly on the discussion of survey answers. Relevant feedback from user interviews will be included in the discussion of a related question.

6.3.1 Pre-Survey

The following paragraphs will discuss the most interesting results of the pre-survey. We'll provide insight to both individual questions as well as overall trends the survey indicates. Six participants submitted the pre survey.

Do you use any location-based social services? Only 2 out of 6 participants stated that they are currently using location-based services like Google Latitude or Foursquare

Usage of Google Latitude 3 participants have never used Google Latitude at all. One has tested but abandoned it, one still uses it several times per week and one participant even uses it on a daily basis.

Usage of Facebook Places Four of our six participants have never used Facebook Places. The other two have tried, but abandoned the service.

Usage of Foursquare The same four participants that have never used Facebook Places have never used Foursquare as well. One participant uses Foursquare on a daily basis. One tried, but abandoned it.

Summary Half of our participants have never used a related LBS at all. One participant tried Google Latitude, but stopped using it. Only 2 of 6 participants are frequent users of location-based services.
This result does not come as a surprise to the authors of the study. In chapter 5, we already described some barriers to the adoption of location-based services. In fact, this reluctance was the main motivation to create Ubiversity, a LBS that limits its scope and puts special emphasis on privacy features.

6.3.2 Post-Survey

After discussing the answers participants gave prior to using our service, we now discuss the result of the second survey in the same way.
Like the pre-survey, the post-survey was submitted by six participants.

Personal Data We first asked the participants some (optional) questions about their age, gender and field of study. All six participants were in the field of electrical engineering; 5 of them students, one a research assistant.
The age of our participants was quite homogenous, ranging from 22 to 25 years with an average of 23.3. One participant was female, five were male.

Checkin Behavior

The first set of questions asked about where and how often the participants checked in. Advanced features like checkins to specific groups or individuals are covered in a later section.

I checked in frequently We asked participants if they felt they checked in frequently. They answered on a Likert scale from 1 (= disagree) to 5 (= agree).
Neither of the extremes (1 and 5) was selected by a participant. The average answer was 2.8 with a low standard deviation of 0.75. All participants used the app to check in from time to time, but no one felt like he used it really heavily.

I checked in frequently at places inside / outside TUM The next two questions asked whether participants checked in more frequently inside TUM or outside, for example at home or while doing sports.
The result is that people used the app to check in at TUM more often than outside of TUM. The average rate of *I frequently checked in at places inside TUM* is 3.5, while the average agree-rate for *frequent check-ins outside TUM* is 2.2. Three participants even completely disagreed (1) with the *frequently outside TUM* statement.

How were your checkins distributed The possible answers were on a 1-5 scale with *1 = always the same place* to *5 = always somewhere else*. The average result is 2.7 with a 1.2 standard deviation. There was a slight tendency to check in frequently at the same place, but the results of this question were quite equally distributed.

Ways of checking in

In 4.1 and 5.10, we described the different localization methods Ubiversity supports: selection from a list, WiFi-based recommendation, NFC tags as well as QR codes. The next four questions asked if people agreed with the statement that they "Checked in frequently using method *xyz*".

Method	Average	Std Deviation	min	max
List Selection	3.8	1.4	1	5
WiFi-Recommendation	4.0	1.1	3	5
NFC Door Tag	1.3	0.8	1	3
QR Code	2	1.1	1	4

Table 6.1: Answers to statement: "I often checked in using *method*"

Clearly, list selection supported by WiFi recommendation was the participants preferred way to check in. However, only very few doors at TUM are currently equipped with NFC and QR technology. Students and researchers that do not work at the VMI group probably never came across such a door.
The results of personal interviews we conducted suggest that especially checkins via NFC door tags would be used way more heavily if more door signs would support this method.

Value of Seeing Friend's Checkins

Before asking about the value a user feels from checking himself in, we first asked three questions about the usefulness of seeing their friends check in.

I like the idea of seeing when my friends are nearby All participants answered either neutral (3) or agreed (4-5). The average is 4.3 with a standard deviation of 0.8. As there are no downsides or costs of seeing where your friends are, this result does not come as a surprise.

By using the app, I met friends more often With this question we were trying to evaluate if participants often actively went to visit their friends after seeing that they checked in somewhere nearby. The result however is neutral. All answers are distributed in the moderate 2-4 range, the average is 2.8 (slightly disagree) with a standard deviation of 1.0.

The system helped to plan appointments and meetings Here, we wanted to find out if the participants already planned using the app when they made an appointment. For example, one could just say *"Let's meet at 3 at the institute"* without specifying a room. The exact room of the meeting could then be shared using Ubiversity.
According to the post-survey however, this case did not happen during the study. No participant selected one of the *agree* answers (4-5). The average was 2.3.

Reason for Sharing Own Location

After asking about the value of seeing the checkins of others, we asked if participants liked to check in, liked when their friends knew where they were. We further asked if they checked in just out of habit, or with a certain purpose.

I like the idea that friends know where I am All participants agree with this statement. The average result was 4.8 with a low standard deviation of 0.4. Of all questions we asked, this is the one with the most unambiguous result.
The interviews we conducted revealed that our participants only added good friends to their friend list. This may be a reason for a *positive anticipation* connected to a checkin. One hopes that a good friend sees the checkin and for example drops by for a cup of coffee.

I shared my location with / without a certain purpose The next two questions asked if participants agreed with the statement that they checked in with, respectively without a certain purpose. The results are quite neutral (3.8 agreement with *with a purpose*, 3.7 agreement with *sharing without a purpose*).

Limitation of Checkin Visibility

Table 6.2 shows how much participants agreed with the statements that they *"Checked in to all friends"*, *"Checked in to a subset of their friends"* or even made their checkins only *"visible to individuals"*.

Visibility	Average	Std Deviation	min	max
Complete Friend List	4.7	0.8	3	5
Selected Group	2.0	0.6	1	4
Individuals	1.8	1.2	1	4

Table 6.2: Participants felt comfortable sharing their location with all people in their friend list.

The participants did not see a need for limiting the visibility beyond their completely friend list. Many of the participants are in general privacy aware, so this can be interpreted that their friend list is limited enough and already gives a good privacy protection.
This impression was supported by the interviews: They only added good friends to their friend list, so there was no need for further limitations. Interviews also revealed that if they had a more diverse friend list, for example with co-workers or lab partners in it, they would use the limitation-features more frequently.

Privacy

The study contains two questions on Privacy: We first asked the participants if they felt that Ubiversity impacts their privacy. In the second question, we asked them to compare Ubiversity's privacy impact with other location-sharing services.

I felt concerned sharing my location Participants generally disagreed with this statement (average = 2). The answers were almost equally distributed with a standard deviation of only 0.6.
This low privacy concerns got confirmed during the interviews we conducted. The friend lists were quite small and homogenous; checkins in Ubiversity are not visible publicly on the web.

I felt less concerned sharing my location than with other services like Foursquare or Google Latitude With this statement, most of the participants agreed. Three completely agreed (5), two still agreed (4) and one participant answered neutral (3).

The answers to the privacy questions are an important confirmation of the overall direction of our work. They support the hypothesis that many Germans are reluctant to use location-based services out of privacy concerns. During the interviews, we identified three properties of commercial LBSs that our participants feel concerned about:

1. Global Scale
2. Connectedness
3. Commercial Interests

The first point expresses that participants feel that services like Facebook Places are too large, not private enough and that they do not want to share their location on a global platform.
The second point, connectedness, means that users of such services often do not know what other services might access their data; possibly in a context they did not anticipate when sharing their location.

The third point expresses the general unease about service providers monetizing personal information, for example for advertising. While personalized advertising is very common in many parts of the web, our participants stated that they do not like their location to be taken into account for such personalization.

Usability

We asked explicitly about Ubiversity's usability.

The app was easy to use One participant did not agree with this statement (2), one answered neutral. Three agreed (4) and one even completely agreed with this statement (5). This result shows the app's usability was probably good enough to not negatively impact the answers to the other questions.
During the interviews, the participants helped us identify most of the usability issues.

This concludes our evaluation of the study. The most important insights the post-survey provided was that the users of the app accepted the concept of a *local web service*. A service targeted at one specific area of life, with smaller scale, and a lower level of connectedness than commercial services.
The results of the study support the claim that such a service has less impact on user's privacy. Furthermore, location-sharing between people that all work or study on the same campus can be more useful than on city-level: Because of shorter distances, users can react to their friend's checkins faster than with more broadly scoped services.

Chapter 7

Conclusion

The contributions by this work include:

- The Identification of essential building blocks of location-based services
- The analysis of these components with respect to the specific requirements of indoor services
- Ubiversity, an example service that combines the developed building blocks into a location-sharing service for students at Technische Universität München.
- A field study that evaluates user's reaction to the *local web service* concept implemented in Ubiversity.

7.1 Reusable Platform

Localization We identified localization features as essential to LBSs (3.1). In 4.1, we discussed the tradeoffs of various localization techniques in the indoor domain. Finally, we implemented a WiFi-based localization-algorithm that recognizes POIs close to a user's location.
The dataset for this WiFi localization can be trained and validated online. This online training happens once a user manually selects his current location. Manual location selection can happen through user-interaction with a GUI, by scanning a QR-Code or by scanning a NFC tag close to a reference location (POI).

Mapping and POIs As identified in 3.2 and in 3.4, location-based services require ways to visualize localization data. Our platform provides tools that support both the creation of floor plans, as well as the setup of a POI database that can be distributed locally on the device or accessed using a RESTful web interface.

Web Backend Our platform for indoor LBS includes a web back-end written on top of the Django web framework. We have described the structure of the backend and its essential concepts in Chapter 3. Most notable among the back-end features is the flexible permission system. It is able to model complex relationships between users and helps to implement the privacy features users expect from modern web services.
We have discussed how to design the backend interface in a RESTful way and gave examples for resource design in 5.3.

7.2 Ubiversity

Ubiversity enables students and staff at TUM to manage friend lists and share their current location. Various ways to checkin are supported, for example WiFi-based recommendation, QR-Codes, and NFC.
Checkins outside of TUM are possible as well: Ubiversity integrates POIs from Google Places or lets users enter their location in a free-text field.

Ubiversity respects the users' needs for privacy and security. Checkin visibility can be limited to groups and individual users; all communication happens securely using SSL/TLS.

7.3 Field Study

In order to evaluate Ubiversity's privacy features and the concept of a local web service, a field study was conducted. The study participants used Ubiversity and gave feedback on their checkin behavior, perceived value, and privacy considerations.

The study results support two claims: First, a location-based service with both limited scope and connectedness, such as Ubiversity, poses a less severe threat to users' privacy than global, commercial services do. Second, a local service still remains useful for location sharing as small distances make an immediate reaction to a friend's checkin more viable than when distances are larger.

7.4 Further Research

The structure of this work indicates the effort put into the design of a general and reusable platform for indoor services. Support tools and documentation were written in a way to enable others to build their own services using the developed framework.

In the future, we plan to put even more effort into the connectedness of our client devices. The support for NFC in smart phones is still in its infancy, dominant usage patterns have yet to emerge. This makes the topic of *NFC as User Interface* an interesting research field. The results could then be integrated into our existing libraries for the Android operating system.

Furthermore, the field study conducted on the Ubiversity location-sharing app supports the hypothesis that local web services with limited scope, for example to a single campus, have less impact on their users privacy than global, connected web services.

The question about the optimal balance between openness and privacy in web services has yet to be answered. It may even change over time, as user experiences and expectations evolve. Privacy-aware web services and the local web service concept will continue to be an interesting topic for further research.

Bibliography

[1] *Django Web Framework*. http://www.djangoproject.org,

[2] *SQLite Website*. http://sqlite.org,

[3] AL., Stefan E.: *NoSQL - Einstieg in die Welt nichtrelationaler Web 2.0 Datenbanken*. Carl Hanser Verlag GmbH & Co. KG, 2010

[4] BRAY, T.: *Extensible Markup Language (XML) 1.0 (Fifth Edition)*. http://www.w3.org/TR/2008/REC-xml-20081126/,

[5] BUTLER, Eric: *Firesheep*. http://codebutler.com/firesheep, October 2010

[6] CORPORATION, Nokia: *Qt Development Framework Website*. http://qt.nokia.com,

[7] CROCKFORD, D.: *The application/json Media Type for JavaScript Object Notation (JSON)*. http://www.json.org,

[8] DIERKS, T.: *The TLS Protocol Version 1.0*. http://www.ietf.org/rfc/rfc2246.txt, 1999

[9] FIELDING, Roy T.: *Hypertext Transfer Protocol – HTTP/1.1*. 1999

[10] FIELDING, Roy T., University of California, Irvine, Diss., 2000

[11] FOUNDATION, Open Street M.: *Open Street Map*. http://www.opentreetmap.org,

[12] FOURSQUARE: *foursquare*. http://foursquare.com,

[13] GUNDOTRA, Vic: *See where your friends are with Google Latitude*. http://googleblog.blogspot.com/2009/02/see-where-your-friends-are-with-google.html, April 2009

[14] HARTER, Andy ; HOPPER, Andy ; STEGGLES, Pete ; WARD, Andy ; WEBSTER, Paul: The Anatomy of a Context-Aware Application. In: *Wireless Networks* 8 (2002), 187-197. http://dx.doi.org/10.1023/A:1013767926256. – ISSN 1022–0038. – 10.1023/A:1013767926256

[15] INC., Apple: *iOS Technology Overview*. https://developer.apple.com/library/ios/#documentation/Miscellaneous/Conceptual/iPhoneOSTechOverview, 2011

[16] INC, Ekahau: Ekahau Positioning Engine 2.0. (2002)

[17] Inc., Google: *Google Places Documentation*. http://code.google.com/apis/maps/documentation/places/,

[18] Inc., Google: *Android Application Fundamentals*. http://developer.android.com/guide/topics/fundamentals.html, 2011

[19] Inc., Skyhook: *Skyhook*. http://www.skyhookwireless.com/howitworks/submit_ap.php,

[20] Jasmin Blanchette and Mark Summerfield: *C++ GUI Programming with Qt 4 (2nd Edition)*. Prentice Hall, 2008

[21] Kreibich, Jay A.: *Using SQLite*. O'Reilly Media, 2010

[22] Krumm, J.: *Ubiquitous computing fundamentals*. Chapman & Hall/CRC Press, 2009 http://books.google.com/books?id=RCxZl4PCXwAC. – ISBN 9781420093605

[23] Likert, R.: *A technique for the measurement of attitudes*. 1932 (A Technique for the Measurement of Attitudes no. 140)

[24] NAVTEQ: *Navteq Website*. http://www.navteq.com,

[25] Richardson, Leonard ; Ruby, Sam: *RESTful Web Services*. 1. O'Reilly Media, 2007

[26] Schilit, B. ; Adams, N. ; Want, R.: Context-Aware Computing Applications. In: *Mobile Computing Systems and Applications, 1994. WMCSA 1994. First Workshop on*, 1994, S. 85 –90

[27] Sharon, Michael E.: *Who, What, When, and Now...Where*. https://blog.facebook.com/blog.php?post=418175202130, August 2010

[28] Stroustrup, B.: *The C++ programming language:*. Addison-Wesley, 2000. – ISBN 9780201700732

[29] Want, Roy ; Hopper, Andy ; Falcão, Veronica ; Gibbons, Jonathan: The active badge location system. In: *ACM Trans. Inf. Syst.* 10 (1992), January, 91–102. http://dx.doi.org/http://doi.acm.org/10.1145/128756.128759. – DOI http://doi.acm.org/10.1145/128756.128759. – ISSN 1046–8188

[30] Weiser, Mark: The Computer for the 21st Century. In: *Scientific American* 265 (1991), September, Nr. 3, 94–104. http://dx.doi.org/10.1038/scientificamerican0991-94. – DOI 10.1038/scientificamerican0991–94. – ISSN 0036–8733