UPTEC IT 10 025
Examensarbete 30 hp
December 2010

Mobile Application Recommender System

Christoffer Davidsson

UPPSALA
UNIVERSITET

Teknisk- naturvetenskaplig fakultet
UTH-enheten

Besöksadress:
Ångströmlaboratoriet
Lägerhyddsvägen 1
Hus 4, Plan 0

Postadress:
Box 536
751 21 Uppsala

Telefon:
018 – 471 30 03

Telefax:
018 – 471 30 00

Hemsida:
http://www.teknat.uu.se/student

Abstract

Mobile Application Recommender System

Christoffer Davidsson

With the amount of mobile applications available increasing rapidly, users have to put a lot of effort into finding applications of interest. The purpose of this thesis is to investigate how to aid users in the process of discovering new mobile applications by providing them with recommendations. A prototype system is then built as a proof-of-concept.

The work of the thesis is divided into three phases where the aim of the first phase is to study related work and related systems to identify promising concepts and features. During the second phase, a prototype system is designed and implemented. The outcome and result of the first two phases is then evaluated and analyzed in the third and final phase.

The prototype system integrates and extends an existing recommender engine previously used to recommend media items. As a part of the system, an Android application is developed, which observes user actions and presents recommended applications to the user.

In parallel to the development, the system was tested by a small group of users recruited among colleagues at Ericsson. The data generated during this test period is analyzed to show the usefulness of observed user actions over explicit ratings and the dependency on context for application usage.

Handledare: Simon Moritz
Ämnesgranskare: Erik Zeitler / Kjell Orsborn
Examinator: Anders Jansson
ISSN: 1401-5749, UPTEC IT 10 025
Tryckt av: Reprocentralen ITC

Contents

TABLE OF FIGURES

1 Introduction

There has been a rapid increase in the amount of information available on the Internet and through other digital media over recent years [1]. This immense amount of information results in an information overload and consumers of information have to put a lot of effort into searching and filtering information.

With over 200,000 applications available for the iPhone and over 170,000 available for the Android platform, there is an apparent information overload emerging in the domain of mobile applications. The fact that these applications usually are browsed and downloaded from a mobile device, with its smaller screen, makes this information overload even more intense.

1.1 Background

In recent years, since Apple's launch of the AppStore[1], there has been a tremendous increase in the popularity and usage of mobile applications. When this thesis project begun there were 80,000 applications available for the Android mobile platform and towards the end, five months later, there are close to 170,000 available [42]. Mobile application stores nowadays do not only offer users an easy way to download applications to their devices, they also provide a shortcut to the market for developers. This has led to a great amount of applications available, for good and bad one could argue. Apple's slogan "there is an app for that" is true in a lot of cases but also leave users stranded with the question "how do I find it".

In order to help users find relevant information, a system could be built to filter out irrelevant or uninteresting information based on what the system believes the user likes. For example, picture yourself in a large city you have never visited, trying to find a good place to eat. There would be lots of options. To reduce the number of options it would certainly help to ask a friend who lives in the city for recommendations, or to consult a magazine for restaurant reviews.

As this example points out, there are different approaches to perform the information filtering. Perhaps search - where the user specifies the criteria for retrieval of information - is the most well known and most commonly used method, but it comes with limitations. With search, a user must know what to look for and where to look for it, and the result of the search is usually not personalized.

[1] Service offered by Apple where customers may purchase or download mobile applications

Another method for identifying interesting products and services is through expert recommendations. Several existing solutions for recommending mobile applications are based on this method. An expert opinion is however only relevant if the user and expert have similar preferences, i.e. expert opinions also suffer from the problem of not being personalized.

With the advent and increased use of social networks in recent years, it is possible to filter products and services based on what a user's friends are consuming. Users could share or recommend applications to each other, a digital equivalent to word-of-mouth, or a user could be allowed to browse her friend's applications. This method results in more personalized recommendations but may introduce privacy issues and not everyone is keen on online social networks.

Recommender systems is another approach to information filtering, which is extensively used in several domains, such as media, products and services. Since the mid-1990s when recommender systems became an important research area, several systems have been deployed. Systems such as Amazon.com have presented recommendations on the form "people who bought this, also bought that" to users for over a decade. A system, which minimizes the information overload in a personalized way would be beneficial not only for the end user, but for every actor in the mobile ecosystem; end users, developers, service providers and retailers.

1.2 Objectives

The goal of this thesis is to build a system, which aids a user in finding mobile applications of interest to her from the first use of the system, at which point the system does not know the user's preferences. The system should also collect application consumption data to base future studies on. The solution should preferably consist of a recommender system and possibly other filtering solutions.

A comparison between application metadata and media metadata from a recommender system point-of-view should also be performed. Suggestions for modifications and enhancements to a recommender engine[2] previously developed at Ericsson AB should be presented.

1.3 Problem

The general problem addressed in this report is the one of presenting applications to the user of a mobile device in a way that minimizes the information overload, i.e. the difficulty a person has to make a decision caused by the presence of too much information. The effort in terms of time and cognitive work a user puts in to finding interesting and relevant applications should be as low as possible. This problem was divided in to sub-problems, namely; collecting and storing application metadata; collecting and storing user consumption of

[2] Recommender Engine is a term used throughout this report to describe a module within a Recommender System with capabilities to produce recommendations.

applications; filtering out less interesting applications; and presenting the interesting ones to the user. These sub-problems are described in greater detail in the following sections.

1.3.1 Collecting Application Metadata

In order to present and filter out applications, the system must be aware of available applications and their attributes. Metadata for each application must be collected and ultimately represented in a generic way in order to be stored and used. There are several platforms for mobile applications (e.g., Google's Android OS, Apple's iOS and the Internet). The applications running on the different platforms are modeled and represented in different ways, and a uniform representation may be needed for future development and to enable generally applicable results.

Collecting the metadata will also differ between the different platforms and how to do this will need to be addressed. Google's Android Market (where most Android applications currently are available) does not provide an official public API. The automatic extraction of metadata applications is a problem, which needs to be resolved.

1.3.2 Collecting Consumption Data

In order to make the application filtering personalized, there must be a user profile. A user profile in the mobile application domain would consist of the user's previous consumption of applications. Consumption could be installed applications; liked applications; applications shared to Twitter; time spent using an application; and so on. To make use of this kind of information it is necessary to know which types of consumption may be obtained from a device and to analyze which types are meaningful.

1.3.3 Filtering and Sorting Information

The general idea of filtering is to obtain a subset of items based on some criteria. In the domain of this thesis, filtering applications is the process of retrieving a subset of applications of high interest and relevance to the user. The resulting subset of applications should be sorted to make applications at the top of the result more likely to be interesting to the user.

1.3.3.1 New Users and Items

Two problems related to recommender systems are the *new user problem* and *new item problem*. Filtering applications for a new user without a consumption history will be a problem since a recommender system will need consumption data to make the filtering personalized.

New items suffer from a similar problem, new items may be ignored (i.e. not recommended) until a substantial number of users have rated the item. These problems are further described in section 2.1.

1.3.3.2 Content

The metadata and attributes associated with applications are not as rich and possibly not as meaningful as the media equivalent. *Appendix A* contains a comparison between application and media metadata and a brief description of the semantics of the metadata. If the metadata of an application does not prove to be rich enough to allow for the content-based approach, a problem may arise. A popular way of solving the new user and item problem has for instance been to use content-based filtering. If content based-filtering is not feasible, the new user and item problem must be solved differently.

1.3.3.3 Context

Mobile devices is a domain where context (e.g. location, time of day, day of week, weather, and so on) may be a factor for successful information filtering. At a conceptual level, this should not cause a problem as the recommender engine to be used for this project has this dimension incorporated in its concept of item consumption. At the implementational level, problems could however form and require an extension or modification in the recommender engine. The recommender engine as of today does not represent context or take it into account when deriving recommendations.

1.3.4 Presentation

Once the system knows which applications to recommend to a user, it must be able to present them. Presentation in this case does not only involve visualization such as the layout of the graphical user interface. Presentation also includes the available metadata for applications, the notion of correlation between applications, the actions one can take on applications, and how this affects the user's workflow.

1.3.4.1 Packaging

How the recommended application is presented on a device greatly depends on what information is made available to the client application. When recommended applications are sent from the server, they will need to be packaged as a set useful to the client application, and in the end to the user. To enable other features than simply listing applications, the set of recommended applications may need to be supplemented by more information when sent to the client. What kind of

supplementary information is useful and how to represent it needs to be investigated.

1.3.4.2 Properties of the Device

The limitations of a mobile device in terms of interaction must be considered when designing the client application. Mobile devices do not only have smaller screens and limited input capabilities, they are also used in environments and contexts different from desktop computers. As a result of this, there is not only a need for a high standard of visual aesthetics; a useful workflow will need to be established as well.

1.4 Hypotheses

Presented in this section are six hypotheses, named H1 to H6. They will be used as a way to evaluate the outcome of the thesis project and will be referred to by their name (e.g. H4). All of these hypotheses are identified as important, if not essential, to the outcome of the system. Except for H4, which is presented out of pure interest.

H1 Mobile application metadata can be gathered.

H2 Mobile application metadata can be represented as a data structure useful for the recommender engine.

H3 A new user without a consumption history can be presented with good recommendations.

H4 A new item, which no users have consumed, can be considered during the application filtering.

H5 Application consumption data can be collected and used to make recommendations.

H6 The existing recommender engine, primarily designed for media, can be used in other domains, such as the application domain.

1.5 Scope

In this section, the scope of the thesis is presented along with a brief description of what will be evaluated and how. The section defines the limitations and focus of the project. The actual evaluation is presented in *Discussion and Analysis*.

1.5.1 Hypotheses

First of all, the hypotheses will be verified or rejected based on argumentation on the behavior and functionality of the system. Some hypothesises may easily be verified by assuring that a part of the system works, while others may require a qualitative study and expert opinion in the form of a discussion.

1.5.2 Modifying a media recommendation engine

An existing recommender engine will be modified and used during the project. Based on the modifications and enhancements needed to recommend applications instead of media, a discussion on modifying recommender engines should be presented. This discussion should evaluate the existing recommender engine, and in a more general sense present a guideline for modifying recommender engines to recommend applications.

1.5.3 Focus

The above sections provide an overview of the focus of this report, which is to investigate how to recommend mobile applications to users of a mobile device. The main, non trivial, problem is how to recommend applications to a first time user of the system.

1.5.4 Limitations

Although the area investigated in this report is highly interesting, it is also very broad and unexplored. A limitation of the scope is needed and therefore the project and report will not focus on the accuracy of recommendations, metrics or scalability of recommender systems. The report will not focus on recommender systems based on social networks or expert reviews.

There will be no thorough usability investigation, no study based on task completion times et cetera. Although a positive user experience is achieved based on functionality, work flow, and interface layout and aesthetics, the focus of this thesis report is on the functionality. Usability aspects will be considered briefly at design time, but not discussed further.

1.6 Disposition

The methodology and approach of the project is presented in Chapter 2. In Chapter 3, recommender systems are introduced and explained in more detail. Different techniques are presented along with a few comments on mobile recommendations versus desktop recommendations and application recommendations versus media recommendations.
The prototype system developed during the project is presented in Chapter 4 and this is followed by a discussion and analysis of the thesis project in Chapter

5. Chapter 6 concludes the thesis and Chapter 7 provides an overview of the future work in the research area and of the prototype system.

2 Methodology

During the project, a state-of-the-art study was performed and a recommender system for mobile applications was built. This chapter gives an overview of and explains the methodology of the project.

2.1 General Approach

The initial focus of the project was to analyze and understand prior research and work done in related areas. Areas such as *information filtering*, *recommender systems*, *mobile applications* and *ubiquitous applications* were studied in depth. Existing solutions attempting to solve problems similar to the one in this report were also analyzed to serve as inspiration and to give an understanding of the effects of certain design decisions.

Based on the findings in the initial research, a prototype system consisting of a server and a mobile client application was designed. The system was then implemented with an initial focus on the server. The client was to some extent designed and implemented in parallel with the server to avoid integration problems early on. Developing the server and client in parallel also enabled an incremental way of working, i.e. there was a working system in place early on, to which features and improvements were added. The prototype system implemented served as a platform for analysis and evaluation of concepts, possibilities and limitations.

The system was evaluated through a quantitative analysis of the outcome of the prototype system and data generated during the tests of the system was analyzed.

2.2 Initial Study

The aim of the initial study was to understand and explore the field of recommender systems in general and to make use of conclusions about previous systems. It was also important to acquire an understanding of how applications differ from other items from a recommender system point-of-view.

As other systems have tried to solve a problem similar to the one presented in this thesis, an analysis of some of these systems was performed (see 2.7 Related Systems). These systems could be viewed as black boxes, as this analysis was limited to observing the system input and output relation. There was no way of knowing what methods were used or any other detail of the systems' design. This analysis is therefore focused on functionality, work flow, interaction and features of the recommendation process apparent from simple output observations.

From the initial study, requirements to guide the implementation phase of the system were derived. These requirements are presented in section 4.1 and in

section 2.7 the related systems are analyzed. The initial study was also the foundation for Chapter 0 on recommender system.

2.3 Proof-of-Concept System

As a way of verifying or rejecting the hypotheses made earlier, a proof-of-concept system was implemented. This system also serves as a foundation for analyzing and discussing the problems and goals set out for the project.

The design and requirements of the system was based on the initial study, and concepts and ideas formulated by me and my supervisor at Ericsson. Before the design of technical details such as classes and data flows, a set of use cases were formulated, which captured more of the desired functionality. When it was clear what would be required of the system, the different modules and their interactions were drawn from which the server interface was defined. Based on the interface of the server, a class design was made along with data flow charts. At this point implementation begun and when the basic functionality of the server was in place, such as handling HTTP requests and data storage, a simple mobile client was made to test the interaction.

2.4 Evaluation

Evaluation of the thesis project was performed through a qualitative study on the outcome of the prototype system, through discussing the hypotheses set out and through a study on user feedback and an analysis of the data obtained during the testing of the system.

The hypotheses set out in the beginning of this report were either verified or rejected through a discussion on the results of the prototype system. Users were testing the system throughout development and their feedback is discussed. The testing also generated data that although sparse allowed for an analysis of application consumption and implicit ratings. The evaluation is presented in Chapter 5, *Discussion and Analysis*.

3 Recommender Systems

Recommender Systems were developed to overcome the problem of information overload by aiding users in the search for relevant information and helping them identify which items (e.g. media, product, or service) are worth viewing in detail. This task is also known as information filtering. This chapter will introduce some recommender system techniques and approaches, give an overview of mobile recommendations, and describe media and applications from a recommender system point of view.

One of the major motivations for a recommender system is serendipity, i.e. to help the user make fortunate discoveries she was not explicitly looking for. Compared to search and expert recommendations, recommender systems also have an advantage in that they are able to make personalized recommendations.

3.1 Techniques

Recommender systems do the information filtering by predicting whether a user will like or dislike an item. This prediction is based on the user's explicit and implicit ratings/preferences, other users' ratings, and user and item attributes. For example, a music recommender could make use of implicit user data (e.g., Sven bought the Beatles' White Album), explicit data (e.g., Sven rated Neil Young 4 out 5), user demographics (e.g., Sven is male), and item attributes (e.g., Nirvana is labelled as Grunge and Rock) to make recommendations.

3.1.1 Content-based

Recommendations can be based on the content of items, comparing the content of previously liked items with the content of unseen items and recommending similar ones. This approach is referred to as the content-based[3] (CB) recommendation method. For example, a system recommending movies would analyze the movies a user likes to find out what they have in common in terms of content, i.e. actors, directors, genres, et cetera. This information will constitute the user's preferences, which are used to find movies with a high degree of similarity to the liked ones [2].

The major drawback of the content-based approach is its inability to identify qualities of items, which are not machine readable or understandable. Humor and visual appeal are examples of such qualities. Content-based filtering also suffers from the *new user problem*, i.e. the user has to rate a sufficient number of items before the user's preferences can be understood [2]. Content-based recommender

[3] Note that the content-based approach should be named metadata-based in these cases, as items are compared based on metadata and not content. Content-based in this paper is an umbrella term for non-collaborative methods that do analyze item metadata or the content of items.

systems also require attribute and feature data of items, and this data may be difficult to collect.

3.1.2 Collaborative Filtering

Collaborative Filtering (CF) recommender systems, on the other hand, recommend items to a particular user based on how other users have rated items. A movie recommender system would find peers, users who have similar rating patterns to the user receiving recommendations. The movies with the highest ratings according to the peers, and which the user has not yet seen, would then be recommended. This approach is called User-Based Collaborative Filtering. There is also Item-Based Collaborative Filtering in which the items a user has rated are compared to all other items in terms of user ratings, the most similar ones with the highest average rating are then recommended.

Collaborative Filtering systems suffer from the *new user problem*, and in addition to this, they also suffer from the *new item problem*. The new item problem causes new items to be ignored (i.e. not recommended) until a substantial number of users have rated the item [2].

3.1.3 Social

Utilizing a user's social network, digital or not, to produce recommendations is in most systems a matter of trust. Trust is a Social Network based recommender system's equivalent to user similarity of collaborative filtering. If a user's trusted connections are known, producing recommendations is a matter of identifying the trusted connections' highly rated items. [39]

The difficulty is, just like in the collaborative filtering case, to identify and rank the user's trusted connections (or neighborhood). But for the sake of argument, one could identify a user's trusted connections as all her friends and friends of friends.

3.1.4 Hybrid

In an effort to avoid the limitations of Collaborative Filtering and Content-based systems, hybrid approaches, which combine the two have been proposed [2]. As an example, such a system could implement the two methods separately and then combine their results. A hybrid approach could also incorporate any other method, such as recommending items based on what has been consumed in locations close to the user.

3.2 Mobile Recommender Systems

Mobile devices, such as phones and PDAs, are evolving in to a major source of information [4]. In the past, many recommender systems have been developed for the desktop computer. However, these systems cannot be applied directly as

an aid for mobile users since mobile recommender systems need to overcome many of the challenges generally present in the domain of mobile devices.

3.2.1 Usability and Interaction

Recommending on a mobile device introduces new challenges due to the properties of the device itself and the context of its use. Compared to traditional desktop computers, a mobile device has a smaller screen and limited input capabilities [8]. The screen size of future devices is unlikely to improve as it is a necessity for the device's mobility. The users of mobile devices will also be in a different environment compared to the desktop. This environment is also likely to be changing continuously during use. Theoretically, any context must be considered when designing the user interface and workflow of the application. The user could for instance be walking around and paying more attention to traffic and other pedestrians than to the device itself and the application running on it.

3.2.2 Visualization

The traditional way to display recommendations is through a list of ranked items in descending order, much like the results of a regular search. Several techniques have been proposed to solve the problem of the small screen when displaying search results on mobile devices, with the typical approach being to display a short description for each item in the result. However, in recommender systems, features of each item (such as category and creator) are usually known, and a set of key features could be identified and displayed to describe the recommended item [4].

Due to the limited screen size, a way to display more information about a selected item is crucial. There should be an efficient way to learn more about an item and as efficient to return to the list of recommendations.

3.2.3 Possibilities

Mobile devices are not all about limitations, they also offer great possibilities. For instance, the physical location of a device and sensory data could prove an important and valuable source of information [4]. This type of data reveals the context in which the device is used at a given time. This contextual information could aid the recommendation process by providing more data to base recommendations on. It may for instance be wise to recommend travel related applications to a user who is located at an airport.

3.3 Recommending Media

Traditionally, many recommender systems have been used to predict ratings for some form of media (e.g. books, movies, music). Media can usually be well described by its metadata. Movies for example, can be characterized by the actors,

directors, genres, writers, production year, cinematographers, and so on. It is often the case that people rate movies by a certain actor or director similarly, making media suitable for content-based recommendations (as well as collaborative filtering, which however is less sensitive to the quality of the metadata).

3.4 Recommending Applications

As noted earlier in this report, there are over 200,000 applications available for the iPhone and over 170,000 available for the Android OS, and no solution addressing the resulting information overload satisfyingly. The general mobile interaction issues discussed above contribute to the need for a way to filter and present the available applications. There has, however, not been much research done in the field of recommending applications or software.

3.4.1 Business Motivation

Recommendations are not only motivated by the user's perceived reduction in information overload. [7] show that recommendations also provide a business value. In their study they compare the difference in sales of mobile applications between users who receive personalized recommendations and users who are only presented with top-lists (such as top-selling applications). "It could be demonstrated that recommender systems are capable to stimulate a measurable increase in overall sales by over 3% on the entire platform." [7]

3.4.2 Application Metadata Compared to Media Metadata

Unlike media, the attributes describing applications are not as elaborate and useful. Applications may belong to a *category*, which corresponds to the genre of a movie. Applications are also described by their creator, installation size, number of downloads, version, etc. However, these attributes are not linked between applications in the same way as actors for movies or musicians for music. For example, it is not likely that a user will like most applications whose version is 1.3.1. This quality of applications might make them less suitable for content-based approaches and is something that will need to be investigated further.

3.5 Humans and Recommendations

It is important to remember what the purpose of producing recommendations when building a recommender system. What it all comes down to is helping human beings make decisions and discover new items with less effort than by manually searching for items. Making recommendations is not only about minimizing or maximizing a metric.

3.5.1 Diversity versus Accuracy

A metric often used to benchmark recommender systems is the accuracy of the predictions produced by the system. The accuracy of a recommender system is expressed as the probability that a user will appreciate the items recommended to him or her [22]. A high accuracy for each recommended item may however not guarantee that the satisfaction with a whole list of recommended items is high. For instance, if many items in a list are too similar to each other, they might not provide any added value to the list.

If a user has rated a couple of movies by Tarantino highly, displaying a list recommending the rest of his movies may not be a good idea. Chances are the user has already heard of, or seen, most of the recommended movies. Although each recommended Tarantino film by itself has a high accuracy, the list as a whole may have been interpreted as more interesting and useful if other movies were presented in it as well.

To deal with this problem [22] and [32] present diversity, a method aiming a making lists of recommendations more valuable and interesting. The idea is to make sure that a list presented to the user contains as diverse items as possible, thus reflecting a larger part of the spectrum of the user's interests and preferences.

3.5.2 Explanations

[17] argues that even though a recommender system mostly makes good predictions, occasionally it will make bad recommendations. As many systems are black boxes from the user's perspective, she is given no indication to consult when questioning whether to trust or doubt a recommendation.

When receiving recommendations from other human beings, one is able to ask the recommending person *why* a recommendation was made. One could then analyze the logic behind the recommendation and determine whether to trust it or not. In the same manner, it would be beneficial for a recommender system to be able to answer this question of why a recommendation was made. According to [17] such techniques have been beneficial in expert systems, which is another type of decision aid.

In [17], explanations in recommender systems are said to give better understandability and acceptance of the system and better control for the user. Users also tend to form a mental model of how they believe the system derives recommendations. By providing users with explanations, an incorrect mental model may be corrected. If such explanations are possible to extract from the recommendation process it would be wise to include them in the systems visual interface, as there are clear benefits for the user.

3.6 Related Work

Previous work in recommender systems has in general addressed issues such as accuracy [1] [18], scalability [1], metrics [19], the cold-start problem [20] and explanations of recommendations [17].

Research directly related to recommender systems for mobile devices have been focusing on visualizing the resulting recommendations [4] [11] [12], [7] has shown the business value of recommendations by looking at mobile application sales, and [5] [6] [9] [10] propose utilizing the context of application use and consumption. Incorporating context into recommender systems is suggested to be done by adding a new dimension, representing the context, to the item-user matrix [9]. This report investigates the use of context by recommendation based on user location.

Many previous mobile recommender systems focus on point-of-interest, travel and tourism, e.g. [4] [13] [16], and media [4] [15]. However, there has not been much published work in the area of recommending games and applications, which is the focus of this report. [5] presents a hybrid recommender system for context-aware recommendations of mobile applications but does not evaluate the system with real users and therefore does not indicate how well different algorithms perform. The system described in this report could be used to gather data for such analysis.

According to [21] explicit user input and ratings should be avoided or kept at a minimum as it requires an effort from the user. It predicts that implicit ratings could be used instead, and that "predictions based on time spent reading (implicit rating) are nearly as accurate as predictions based on explicit numerical ratings". Implicit ratings have also gained in popularity since the article was published in 1997. This system of this report is soley dependant on implicit ratings.

As [22] points out, a user's level of satisfaction with a recommender system is not only based on the accuracy in the recommendations. To provide serendipity and capture the broader taste of a user, [22] suggests diversifying recommendation lists and introduces a metric for this purpose. Another way to increase user satisfaction is by explaining recommendations, a topic addressed in [17]. It suggests that explanations increase the user's trust in a system and acceptance thereof. Explanations could also educate the user as she will get a better understanding of the strengths and weaknesses of the system.

3.7 Related Systems

In this section some related existing systems are studied as reference and inspiration. This was done to acquire knowledge on the strengths and drawbacks of other solutions and to establish a ground to base design decisions of the Mobile Application Recommender on. Note that this investigation does not analyze or comment on the algorithmic solutions, the focus is rather workflow and feature oriented.

3.7.1 Appazaar

Appazaar [34] is a context-aware system for recommending mobile applications available for the Android platform as a prototype implementation. In their paper, Böhmer et al. explore the design space for a system similar to the one presented as a prototype in this report.

The novelty of mobile applications as items in context-aware recommender systems is highlighted due to the importance of context within the mobile domain and the possibilities to continuously capture application usage along with contextual information. The Appazaar mobile application uploads observed user actions to a server, which in turn sends recommended applications to the mobile application, where they are presented.

3.7.2 AppAware

AppAware [33] aims at aiding users in making fortunate discoveries of mobile applications by making use of context (location) to produce recommendations. AppAware provides the user with a continuous stream of application related events (installs, uninstalls and updates) to make the user aware of what other people are consuming in her proximity.

The system's main target group is frequent travelers interested in discovering applications useful at their specific location. The system does not make use of any common recommender techniques, such as collaborative filtering or content-based filtering. Recommendations are solely based on the user's location and application consumptions in the vicinity of this location.

3.7.3 Appsaurus

Appsaurus is an application for the iPhone platform and is run on the device to serve the user personalized recommendations of applications [26]. As shown in Figure 1, Appsaurus presents the recommendations as a list of five items from which the user selects his favorite one by clicking it resulting in a new list. From each list, the user can also choose to view more details, such as a description and screenshots, for each application, or to place the application in a list, such as *blocked* or *favorites*. It is also possible to create custom lists. From the details of an application, the user can choose to download or buy an application, which redirects the user to the application in Apple's AppStore.

From the beginning, when the user is new, the applications presented are more or less randomly chosen, but as Appsaurus gains more knowledge about the user's preferences the recommendations become more personalized. Since the user is encouraged to choose his favorite application in each step, Appsaurus always receives feedback to derive preferences from. The user on the other hand, is always presented with applications and although none of the recommended ones are considered download-worthy, simply clicking on the most interesting one will result in a new list of recommendations. This seems to build an efficient flow of recommendations in an interactive way, beneficial to both user and system.

Figure 1. Appsaurus screenshots.

As shown in Figure 2, it is possible to create custom filters to receive recommendations within a user specified price range or in a specific category.

These are some of the positive Appsaurus features noted, and although the application is really well made overall, there are a few flaws. First, and most important, Appsaurus does not identify which applications the user has installed and therefore recommends installed ones. The user has to explicitly place an installed application in an *installed* list. Secondly, Appsaurus does not make use of the context in which recommendations are made. Recommendations could for instance include applications popular in the general vicinity of the user.

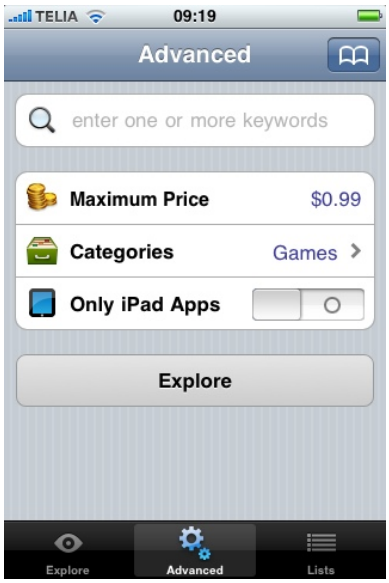


Figure 2. Appsaurus' custom filters.

3.7.4 AppStore

AppStore by Apple is the official way to download applications for the iPhone. The user can browse iPhone applications by category, by a list of applications, featured applications, most downloaded and search for applications. AppStore also comes with a recommending feature named Genius. Genius explains each recommendation with a short message such as "Based on Application X", see Figure 3. As explanations could

be helpful to the user and build user trust in the system, it is important to make it right, otherwise the user may simply be confused. In AppStore's Genius the recommendations may be dismissed and there is a button for retrieving more recommendations.

3.7.5 Appsfire

Appsfire is a website and native application for discovering and sharing applications [23]. At the moment Appsfire is targeting the iPhone platform. Appsfire lets the user discover applications online by browsing lists of applications made by experts and users, and through a top-list of the most popular applications on Twitter. However, more interesting to this report is the iPhone application companioning the Appsfire website[4]. The iPhone application allows the user to get personalized recommendations presented on the device, in addition to the features available online. Figure 4 shows the application in *discovery mode* where the user can choose "Suggested Apps" at the top to receive recommendations based on personal taste, also shown in Figure 4.

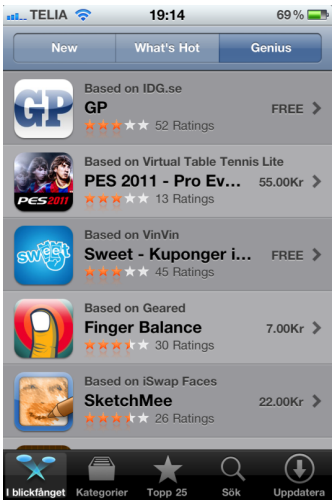
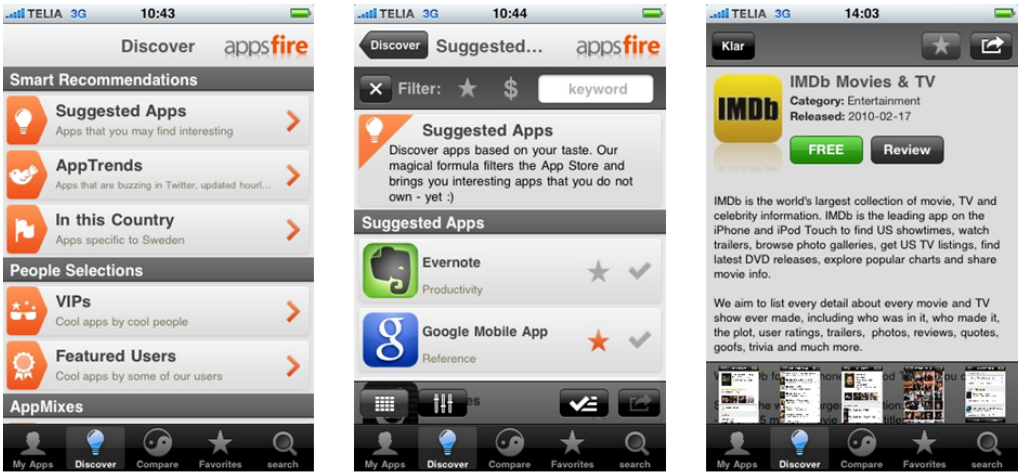


Figure 3. Genius in AppStore iPhone application screenshot.



[4] http://www.appsfire.com

Figure 4. AppsFire iPhone application screenshots.

Some features of the application interface and functionality are considered noteworthy and desirable. In a list of applications, clicking a star to the right will add the application to the user's favorites, and a check-box enables the user to share checked applications to social networks, by e-mail or SMS.

When an application has been selected and is displayed to the user, screenshots of the application are available at the bottom of the screen and are brought up to full screen with a click. The applications a user has installed are browsable and sharable. Furthermore, the applications may be made favorites in the same fashion as non-installed applications.

On the downside, the list of recommended applications is limited in size. If a user does not like any of the recommendations, there is no way to dismiss the recommendations or get more. Also, in order to receive personalized recommendations on the device, a desktop application must be downloaded and run to import installed applications, i.e. it is not possible to simply download the application from AppStore on to the device and instantly get personalized recommendations.

3.7.6 AppSpace

AppSpace [24] supports several platforms (iOS, Android, et cetera) and is a web based service, which recommends applications to the user. Based on the user's installed applications and ratings of applications, AppSpace will recommend applications the user might like. A major drawback of AppSpace is that the user has to explicitly inform the system of his preferences such as installed applications. However, a good feature is the use of lists, as indicated by the bottom arrow in Figure 5. New lists can be created and named. Applications can be placed in any number of lists, for reference or as an archive. This could be very useful in a mobile context to enable users to save apps for later consideration in an archive. Lists could also be a way to obtain more metadata about an application, placing an application in a list could be seen as tagging the application with the list name.

AppSpace also gives feedback by letting the user know how good recommendations to expect, indicated by "Discovery Strength" as highlighted by the top arrow in Figure 5.

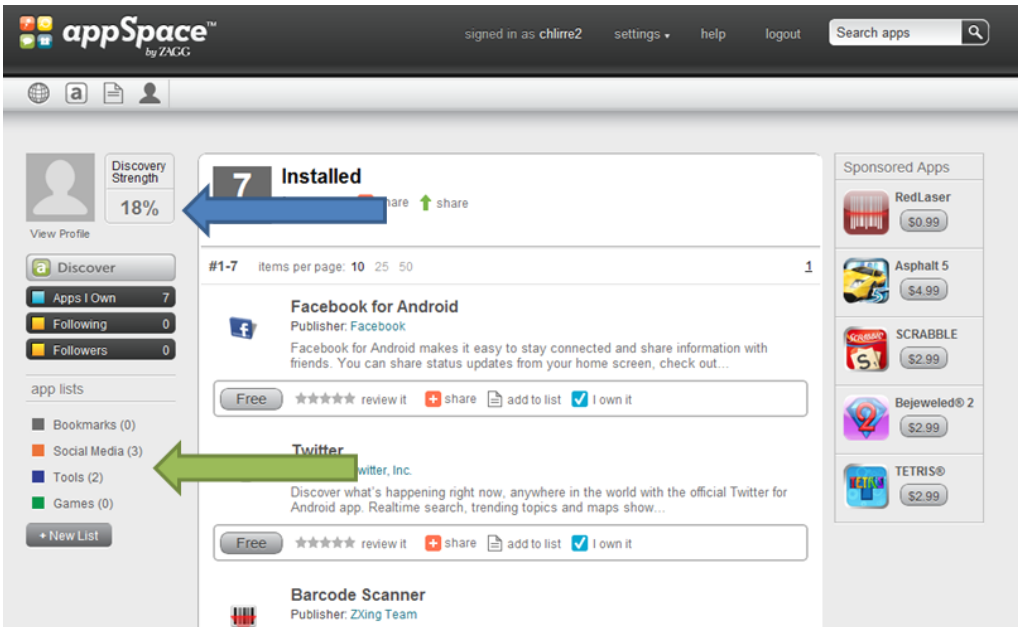


Figure 5. The AppSpace website.

4 Mobile Application Recommender System

This chapter describes the system prototype. The requirements (4.1) are specified, tools and frameworks (4.2) are presented and the prototype system described (4.3).

4.1 Requirements

Based on the goals set out in section 1.2, the findings during the initial research study and the above analysis of existing systems, a set of requirements can be formulated. This section divides requirements into two categories; weak and strong. Weak requirements are desirable features and qualities of the system, whereas strong requirements must be accomplished by the system.

4.1.1 Strong requirements

- The system should be able to collect application metadata
- The client of the system should be able to detect applications installed on a device
- The system should collect usage data from the user's device
- Contextual information should be associated with usage data
- The system should, based on the usage data, recommend applications to the user
- The system should use an existing recommender engine to derive the recommendations
- The system should allow the user to install recommended applications

4.1.2 Weak requirements

- The system should consider context when making recommendations
- The system should allow the user to place applications in lists
- The system should minimize explicit user feedback

4.2 Tools and Frameworks

This section will introduce the tools and frameworks used throughout the thesis project to build the prototype of the Mobile Application Recommender. Developing for the Android Platform and making use of the Ericsson Recommender Engine were requirements set out by Ericsson. Using the Eclipse Integrated Development Environment (IDE) to aid the development was a recommendation from Ericsson. Since development for the Android platform is carried out in Java, the natural choice of programming language to implement the server side was also Java and therefore Java Enterprise Edition (EE) has been used to implement the server. This section carries on with focus on the Android platform and the Ericsson Recommender Engine.

4.2.1 The Android Platform

To implement the system, the Android platform was chosen for the client side. Android is an open source platform and operating system for mobile devices based on a Linux kernel [28]. There is a Software Development Kit (SDK) with all the tools and APIs necessary for development [27]. Applications are written in Java and run in their own process with their own instance of a Virtual Machine. Android was initially developed by Google but is currently part of the Open Handset Alliance [29].

In Figure 6, the Android software stack is shown, it illustrates that applications can make use of the application framework for standard Graphical User Interface (GUI) components, various managers such as the Activity Manager and Content Providers for storage.

There are four major components to an Android application; Activities, Services, Broadcast Receivers and Content Providers. An Activity presents the GUI and is given the full size of the display to draw in. The Activities also handle user touch input. A Service does not have a user interface, but is run in the background, and could for instance be fetching data over the network. A Broadcast Receiver does nothing but, as the name suggests, receive and react to broadcast announcements sent out by the system or other applications. Content Providers store and retrieve data to make it available to all applications, and are the only way to share data across applications. Content Providers are also the preferred way to handle data storage within an application and was used in the prototype system of this report, SQLite is the database used by the Content Providers.

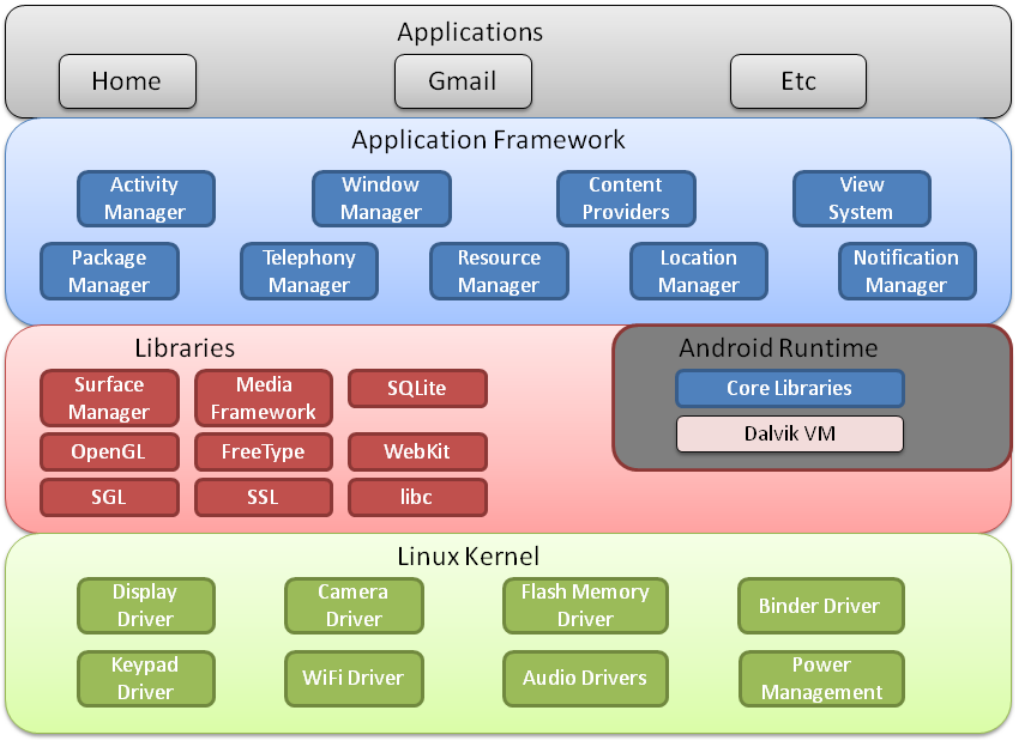


Figure 6. The Android Platform Architecture.

4.2.2 Ericsson Recommender Engine

A recommender engine was developed by Ericsson prior to this project. It can be seen as a black box in which item data and user consumption and ratings of these items are stored. Upon request, the recommender engine will return recommended items for a specified user. It was designed to be used as a separate service, and has previously been used to recommend media. The system proposed in this report will use the recommender engine to produce application recommendations. Several algorithms such as user-based and item-based collaborative filtering, content-based and combinations of these are implemented in the recommender engine.

4.3 Prototype System

The Mobile Application Recommender system implemented prototype is presented in the following sections. The prototype is not complete in functionality or design, but serves as a proof-of-concept and is used as a basis for conclusions, evaluation and discussion in this report. An overview of the system design is presented first, followed by a more detailed description of the client and server. The server and client sections describe the system parts as they were implemented during the project, the ideal system and concepts developed but not implemented during the thesis project are discussed in *Future Work*.

4.3.1 System Design Overview

As a fundamental requirement, a client application was to be running on a mobile device and Android was the chosen mobile platform. Apart from the Android application running on a mobile device and the existing recommender engine, the system also consists of a server. An overview of the system design is shown in Figure 7.

In this design, the client is responsible for collecting and sending consumption data to the server. The client makes requests for recommendations to the server and displays these recommendations to the user. The server collects application metadata, stores consumption data received from clients, and forwards requests for recommendations to the recommender engine and sends the response back to the client. The client and server are described in greater detail in the following sections.

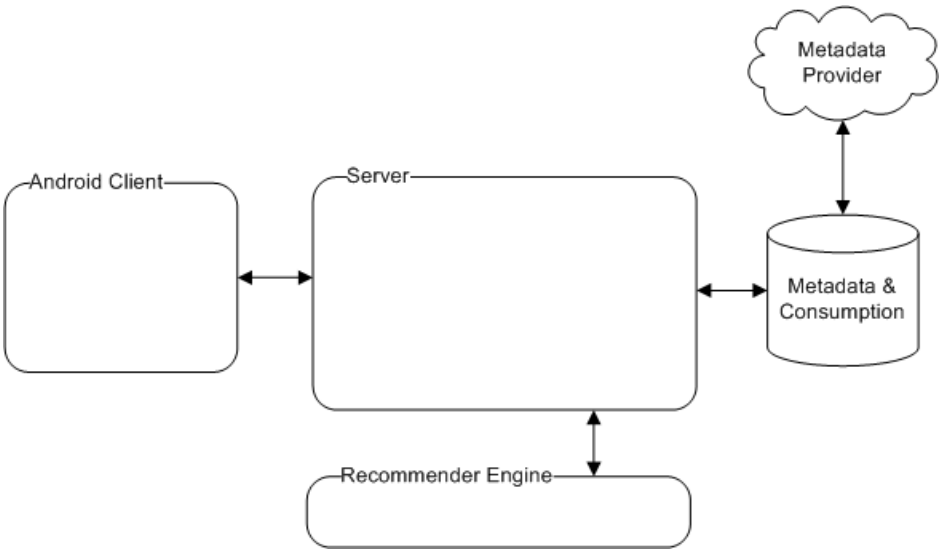


Figure 7. System design overview.

The motivation to this design is that although the client could make requests directly to the recommender engine it would have to store lots of data, such as consumption data and application metadata. The client could possibly even have the recommending functionality built in. However, the computational power and storage capabilities of mobile devices is limited. There is also a convenience in having a centralized way of collecting application metadata. The task of collecting and managing applications in the system does not need to become a distributed problem, and the recommender engine can remain unchanged, as it is not part of the collecting. This solution also allows modifications to parts of the system, without a need for the client to be updated by the user.

Any recommending functionality desired but not implemented in the recommender engine was added as a layer on the server side. These extensions could serve as input for the future development of the recommender engine and give an insight into how recommending applications differs from recommending media.

4.3.2 Client

To describe the client application of the system, a task based approach has been taken. The following sections will describe how the client is used and the inner workings of the client, using a top-down approach.

4.3.2.1 Work Flow and User Interface

To give an overview of what the client looks like and what it does, this section presents the user interface and work flows. The typical use of the system is shown in Figure 8, where the user is first shown the *home screen* of the application (a). If the user then presses "Discover", she is shown a list of recommendations (b). From the list of recommendations the user may view an application's details (c) by pressing an item in the list, or display a pop-up menu with available actions (b).

Figure 8. Application screenshots.

Other features of the client application are presented in the following sections and are shown in Figure 9 and Figure 10.

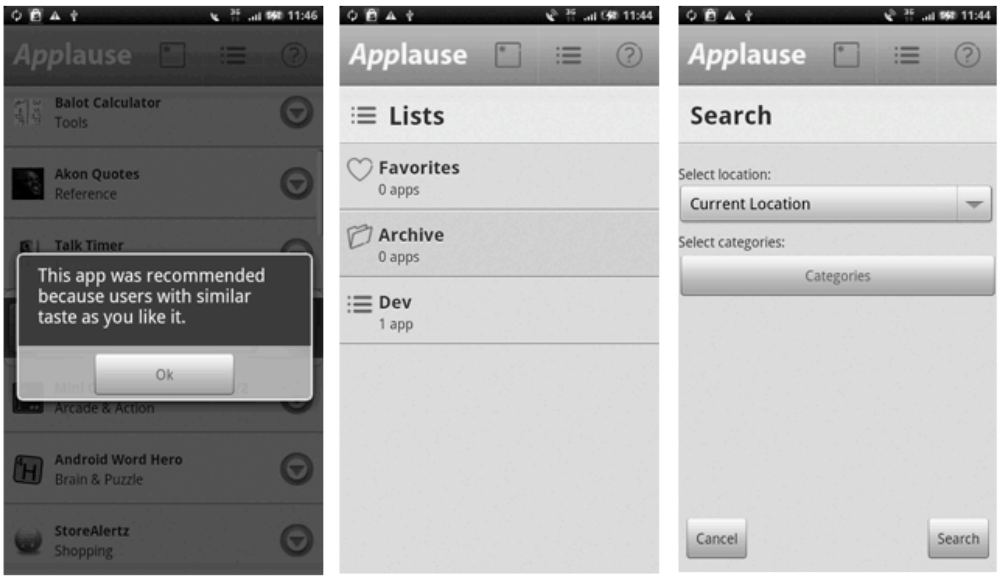


Figure 9. Application screenshots.

4.3.2.2 Collecting Consumptions

As recommendations are presented and applications are run on the same system as the client application of this prototype system, it is possible to monitor the user's actions. Access to contextual information such as the time and the user's location is also made available through the Android platform and this information can therefore be associated with each consumption.

Table 1 shows the types of consumptions collected by the system and their meaning or related action. If an application was *consumed*, it was run on the device in the foreground. A consumption of the type *save* means that the user chose to install the application, and the opposite to this is *delete,* which corresponds to an application being uninstalled. A *glimpsed* application was recommended and shown to the user, whereas *associate* only means the application information was sent to the device as a recommendation. Hence, *glimpsed* is a subset of *associate*. If a user views the details of an application, it is stored as a consumption of type *consider*. Users are enabled to place applications in lists within the client application and such an action is referred to as a *mark* and a request for similar applications for an application is of the type *query*.

A consumption is made up of the user's id, the application's id, the context in which the consumption took place and the type of the consumption.

Type	Meaning
Consumed	The user ran the app in the foreground
Save	The user installed the app

Glimpse	The app was shown on screen in a list (of recommendations) to the user
Consider	The user viewed the details of an app
Mark	The user placed the app in a list
Associate	The app was returned as a recommendation to the user (but was not necessarily display on screen)
Query	The user requested apps similar to the app
Delete	The user uninstalled the app

Table 1. Consumption types collected by the system and their meaning.

The context contains information on the time and location associated with each consumption. This could be extended to include any information describing the context, such as battery status, sensory data, network connection, other applications running, et cetera.

Some of the consumptions, namely glimpse, consider, mark, associate and query, are logged from the Activities (the views) of the client application. The remaining consumption types are logged in a Broadcast Receiver, which listens for broadcasts sent by the system, notifying its listeners of events. Each time a consumption is detected, it is stored in an SQLite database.

4.3.2.3 Posting Consumptions

As the consumptions are stored locally rather than sent to the server instantly after their creation, they can be retrieved from the database and sent at any time. This allows for freedom to modify when consumptions should be sent. It could be done upon request from the user, when a request for recommendations is made, timer based, as soon as the user has Wi-Fi access and so on. In the current implementation consumptions are sent to the server when the application is launched. They are also sent when the number of consumptions stored in the database reaches a certain number, to avoid them taking up too much space in case the user decides to stop using the application.

The server has a REST interface, which accepts json-string [38] representations of consumptions. The stored consumptions are converted to such strings with the use of Gson, a Google made Java-library for converting Java objects into their json representation [37]. A list of such strings is sent to the server using the org.apache.http library, which is part of the Android platform.

To save storage space on the mobile device, consumptions are deleted once they have been sent to the server. They are not deleted until the server has responded that everything went fine and that the consumptions have now been stored server-side.

4.3.2.4 Retrieving and Presenting Recommendations

This is the main task of the system. It is where the user will evaluate whether the goal is met or not. The other parts of the system are in place to make it possible for the user to request and browse recommended applications.

Requesting recommendations can be performed in two ways in the client application, the user can either make a general request or use the search functionality to specify some criteria for the request. This selection is done on the home screen (Figure 8) of the application, where pressing *Discover* results in a general request and *Search* takes the user to the search activity shown in Figure 8 (c). Discovery can be reached by pressing the discovery-icon in from the menu-bar at the top of the application as well.

From the search activity, the user is enabled to specify a location on which the recommendations should be based. To select a location, the user is shown a zoomable and movable map (Figure 10(c)), and by clicking a spot on the map for more than one second makes the location selected. This could be useful in case the user is going on a vacation and wishes to download applications relevant to the destination before the trip is made. It could also serve as inspiration to know which applications are popular in New York or at the NASA headquarters. From the search activity, the user may also select application-categories, which make only applications from these categories valid for recommendation.

A request for recommendations is communicated to the server in the same way as the consumption in the previous section. A json-string representation of the request is generated using Gson, and it includes information about the location of the request, the user's id, categories and a boolean indicating whether the recommendations should be based only on location or not.

The response from the server to such a request is a list of applications, in the json format, which is transformed in to a Java list of Application objects using Gson. The presentation of the recommended items is the same for the two types of requests and is show in Figure 8(b). A scrollable list is used to present the recommended applications and includes information such as icon, title and category. Pressing the arrow to the right in each list item reveals a pop up-menu with the possibility to install the application, place it in a list, request similar applications or get an explanation as to why the application was recommended.
The applications are ordered according to a rank provided by the server. Each application has an attribute indicating the applications position, where *1* is the most relevant and should be displayed first in the list, *2* is the second most important and so on.

Received applications are cached on the device until the next request for recommendations is made. They are cached to allow the user to accept a phone call perform other tasks, without recommended applications disappearing. The cached applications are removed as new ones are received to save storage.

By pressing a list item itself, the details (as shown in Figure 8(c)) of the application are shown and the user may take actions such as installing the application or placing it in a list. If the user decides to install the application, it will be launched in the Android market on the device where the user needs to confirm the installation before the application is downloaded.

4.3.2.5 Placing Applications in Lists

In the above section, it was mentioned that applications may be placed in a list. What this is and how it is done is described in this section.

Application-lists is a feature which enables the user to archive or manage applications. An application may be placed in a list for inspection at a later time, if the user for instance wants to wait until she has Wi-Fi access, before downloading the application. Lists also allow the user to group applications, for application management purposes. An idea for the future is to enable users to install or uninstall all of the applications in a list in one click. This is however out of the scope of this thesis and has not yet been implemented.

Two lists are predefined as the client application is installed; Favorites and Archive. When placing an application in a list, the user may however create her own lists. This is shown in Figure 10(a) where the user may select an existing list or create a new one. By selecting "Lists" from the home screen the user is shown the available lists, and pressing one of the lists shows the applications placed in this list as shown in Figure 10(b) where the contents of the *Archive* list is displayed.

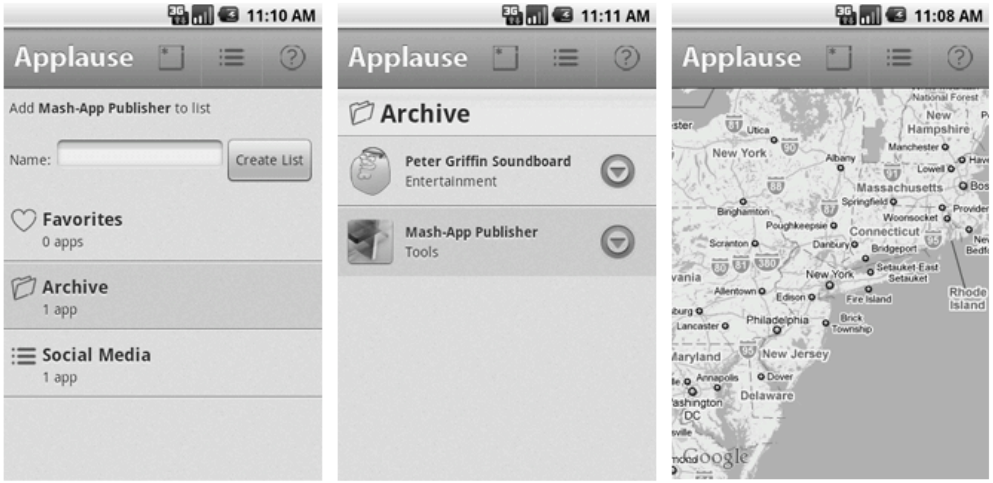


Figure 10. Application screenshots.

4.3.3 Server

This section takes a top-down approach to describe the inner workings of the server. The most important features of the server are presented to provide an overview of what the server does.

4.3.3.1 Functionality

As apparent from the previous section on the client and from the requirements in section 4.1, the server must be able to receive and store consumption data,

receive a recommendations request from the client and respond to this request by computing recommendations. To provide this functionality the server must also be able to collect meta data for applications.

A RESTful web service is a simple web service for accesssing resources through HTTP in a stateless manner. Stateless meaning that each request from any client contains all of the information necessary to service the request.
The communication features of the server are implemented through a RESTful web service, which is described in the following section.

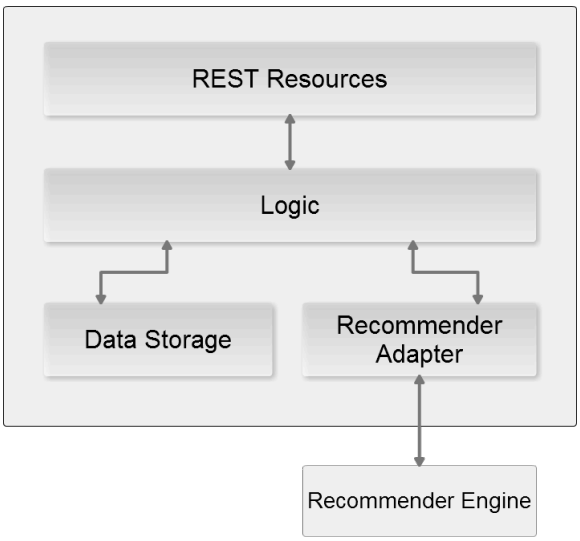


Figure 11. The conceptual layers of the server and their interactions.

4.3.3.2 REST Resources

As seen in Figure 11, the REST resources are the top most layer of the server where they serve as an interface to the underlying functionality. Each request made to the server is handled by the REST resources and forwarded to the appropriate part of the logics layer.

Each resource presents an interface to an entity such as application, recommendation, consumption or user. A request to the server is made through a HTTP request and results in a call to the appropriate method of the class representing and implementing the resource.

When a request has been processed by the logics layer, the REST layer creates a response, which is sent to the requesting client. This response is either empty with a status code indicating whether the request was successful or not, or contains a message representing the requested resource. For instance, a request for recommended applications returns a list of applications as a json-string.

4.3.3.3 Logic

Once requests to the server have been interpreted and any additional data has been parsed by the REST layer, the request is passed to the logics layer, which takes the appropriate action. The request to put consumptions, from the previous section, is pretty straight forward. The appropriate action is to store the consumptions in the database.

A request for recommendations is passed on to the recommender engine adapter, which generates recommendations based on the incoming request.

The logic layer makes sure to instantiate any objects needed by the underlying layers and passes them to appropriate methods.

4.3.3.4 Data Access and Storage

Data persistence is handled using Hibernate, which is a persistence framework for Java EE. Through an Object/Relational mapping, Java object are fetch and stored. A mapping file was written for each entity defining the mapping from Java object to its relational database representation. A configuration file defining which database to use and authentication information for this database is defined and used by Hibernate to connect to the database.

The database used in this thesis is a MySQL database. Appendix B shows the model of the database. Each of the entities in this model has a corresponding Java class, which it maps to. Using Hibernate and MySQL allowed for a rapid implementation of the persistence functionality.

4.3.3.5 Recommender Engine Adapter

A recommender engine adapter was implemented to serve as layer between the server and the already existing recommender engine. This layer formulates requests for recommendations, puts data to the recommender engine and makes requests for the recommender engine to build its models. Additional recommendation features, such as location based recommendations and filters were also built into this layer.

Data, such as consumption data and application metadata, is sent to the recommender engine prior to build requests. Build requests are made by an administrator to the REST resources described above. When the models of the recommender engine have been built, recommendations can be requested for. The recommender engine provides recommendations based on collaborative filtering, both item- and user-based.

The system provides recommendations based on the user's location, and as location-based recommendations are not implemented in the recommender engine, this functionality was built in to this layer of the server. Given the geographical coordinates of the request, recommendations are produced by retrieving application consumption within a radius of this location. The retrieved

applications are ranked based on distance and consumption type. An application consumed close to the requesting location scores higher than an application consumed farther away. An application with consumption type *save* scores ten points, a consumption of type *consume* scores five points and a consumption of type *mark* and *query* scores one point. The applications are then sorted to present the applications with the highest score first.

4.3.3.6 Meta Data Collection

For the server to be able to send applications with metadata such as title, version and so on to the client in an efficient way, this information needs to be stored server-side. As the Android platform was used in this project, Android application metadata should be collected.

Google does not provide an official API to the Android Market, which is where almost all applications are available from to Android users. There is however a third party open-source API available [41]. This API was used on the server to fetch application metadata.

The collection of metadata is instantiated by an administrator through a HTTP request, which could be performed from any browser or other HTTP client. Table 2 lists the different admin requests available and their meaning.

Admin Request	Meaning
Collect Random	The server fetches application meta data and random
Collect New	The server fetches application meta data for applications marked as ***new*** in the Android Market
Collect Featured	The server fetches application meta data for applications marked as ***featured*** in the Android Market
Collect Popular	The server fetches application meta data for applications marked as ***popular*** in the Android Market
Maintain	The server identifies applications with missing metadata in the database, and fetches metadata for these applications

Table 2. Meta data admin requests.

Applications are run client-side and their identifying string sent to the server as they are consumed. If a consumed application is not already present in the database, its identifying string will be stored but the other metadata fields are left blank. The maintain request of Table 2 should be used to populate the metadata of such applications.

5 Discussion and Analysis

This chapter discusses and analyses the result of the project, evaluates to what extent the problems presented in section 1.3 were solved. An argumentation for rejection or verification of the hypothesis of section 1.4 is presented and the methods applied during the project are examined and evaluated.

The evaluation will mostly be performed in a qualitative manner as there is little real data to perform tests on and there was no time or plan to perform user testing or surveys. The problems and hypothesis were formulated to allow for expert opinions rather than data analysis. However, the last section of the chapter presents the data that was obtained, to show possibilities for future evaluation in the event of more extensive data collection.

5.1 Hypothesis Evaluation

In this section, the hypotheses will be verified or rejected based on argumentation on the behavior and functionality of the system. The hypotheses, as presented in section 1.4, are stated below and each one is followed by a discussion.

H1 Mobile application metadata can be gathered.

For the Android platform, which was the chosen test platform of this thesis project, this hypothesis is **verified**. Metadata for applications was collected from the Google servers through a third party implementation of the Android Market API. Application portals or stores, which may run on a mobile device needs to utilize a web service for application metadata, it should therefore be possible to obtain this data for other platforms and portals as well. An indication that this is possible for the iPhone platform is applications such as Appsaurus (section 2.7.3) and AppSpace (section 2.7.6), which have iPhone application metadata.

H2 Mobile application metadata can be represented as a data structure useful for the recommender engine.

For a recommender engine to use collaborative filtering to derive recommendations, the only data required for items, such as applications, is an identifier. And thus, for this method the requirement is met. Each application on the Android platform could for instance be uniquely identified by its package name (e.g. com.google.maps). Assumed that applications can be uniquely

identified, mobile applications can be represented to be useful to a recommender utilizing collaborative filtering and thus the hypothesis is **verified**.

H3 A new user without a consumption history can be presented with good recommendations.

First of all, a good recommendation in this case does not necessarily mean a personalized recommendation. Based on the experience with the mobile application prototype, utilizing the location to derive recommendations to new users has been promising. AppAware (presented in section 2.7.2) only uses the user's location to derive recommended applications and [33] identifies a promising serendipity effect of this type of recommendations. If the main focus of a new user is to get started using applications and to catch up with the surrounding application consumptions, then basing recommendations on location is a promising solution. The method of displaying locally popular applications should generate more satisfaction compared to displaying random applications, but may not prove to be better than displaying globally popular applications. Whether to **verify** or **reject** this hypothesis is a matter of defining a good recommendation. The prototype system shows promise in being better than random, but if the user expects something truly personal, the methods presented in this report may fail.

H4 A new item, which no users have consumed can be considered during the application filtering.

Verifying this hypothesis would require a content-based approach, as this is not possible using collaborative filtering. Using metadata to predict items the user will like should enable a solution this problem. Since the recommender engine used to derive recommendations does not provide content-based recommendations as of yet, this has not been tested during this project. In the current setup, the hypothesis is **has not been tested**, but it may be shown plausible by future development.

H5 Application consumption data can be collected and used to make recommendations.

On the Android platform, possibilities to collect implicit application consumption data proved to be very good. This hypothesis is therefore **verified**. Chances are that this may not be possible in all platforms. Observing user actions in web browsers for collecting consumption of web applications is for instance not possible to the same extent. In such cases, relying on explicit ratings may be required.

H6 The existing recommender engine, primarily designed for media, can be used in other domains, such as the application domain.

The recommender engine delivered satisfying results for applications, considering the limited data available. Some modifications of consumption interpretation were needed to provide application recommendations. These modifications were however minor and the hypothesis is **verified**.

5.1.1 Hypotheses Summary

To conclude the discussion on the hypotheses presented above, the results from the project could verify all hypotheses except for H4. This positive outcome of the hypotheses is closely related to the turn out of the thesis project, as the hypotheses represented a basic set of requirements for a mobile application recommender system.

5.2 Extending a Media Recommendation Engine

This section will discuss how to extend the existing recommender engine at Ericsson. A straightforward way of doing this is by identifying which functionality had to be added on top of the recommender engine. From this added functionality, it is possible to identify what should be included in the recommender engine.

The most notable feature which had to be added on top of the recommender engine was exclusive filters. As of today, the recommender engine only supports inclusive filters, i.e. filters where one declares which categories of items or actual items are valid for recommendation. For instance, "give me recommendations for user X from the categories *Action* and *Comedy*". Adding exclusive filters enabled the system to filter out applications which a user already had installed and would allow exclusion of applications in a certain price range. Such a filtered request could be formulated as "give me recommendations for user X, but **exclude** application *Y* and *Z* and all applications with a price higher than $4".

A second feature, which was built on top of the recommender engine is the location-based recommender functionality. Incorporating contextual features into the recommender engine would not only be beneficial for application recommendations, but for any item recommendations.

Diversity - making sure recommended items are not too similar – was not implemented during this project, but the concept was identified as interesting for application recommendations. It is a desirable feature, which should be optional when requesting recommendations from the recommender engine. For instance, a request could include a parameter indicating the level of desired diversity.

The recommender engine of today is model based, and the models need to be built before recommendations can be derived. The major drawback from this is that even if a new user generates a sufficient amount of consumptions rapidly, the recommender engine models have to be rebuilt before the user can receive recommendations from the recommender engine. However, the models may not be built very often, as this is a time demanding process. To enable recommendations for a new user with a sufficient consumption history the recommender engine should be able to take this user's consumptions as input, and from this derive recommendations even though the user is not part of the model.

5.3 User Feedback

The Android client application of the system was distributed to colleagues at Ericsson and fellow students at Uppsala University. A total of 33 users installed the application during an eight week period, and their feedback and an interpretation thereof is presented in this section.

Some of the feedback received was on the visual aesthetics of the application user interface, but this was not within the scope of the thesis. Aesthetic feedback was therefore noted but not further analyzed or considered.

Since the user location was determined using GPS, and this was done constantly in the background of the application to monitor user actions, users complained of high GPS usage, which had negative effects on the battery power. This issue was handled by minimizing the GPS usage. There is built-in functionality in the Android framework allowing an application to receive GPS data only when the device has moved a certain distance or after a desired period of time. This functionality was used to minimize GPS usage.

Users were also wondering how the recommended applications were ordered. The main concern was that there was no notion of the importance or relevance of a recommendation. This issue was solved to some extent by adding explanations to the recommendations. This allowed a user to see why an application was recommended. It would however be desirable to add a relevance measure in the list of recommended applications.

The users also experienced that the system recommended applications that they already had installed. First of all, this indicates that the recommendations were good since the user had already shown an interest in that application and installed it. However, as users pointed, it is desirable only to recommend new applications to the user. In the current setup, already installed applications are filtered out server-side. In some cases however, the server had not yet received and stored information about which applications a user had installed, before the user requested recommendations. As a result, the already installed applications were not filtered out. A solution to this would be to filter recommended applications in the client application as well.

5.4 Data Analysis

During the brief test period, data was collected and an analysis of this is presented in this section. As the number of users was only 33, this data is not enough to draw general and statistically significant conclusions. 8,240 applications in 22 categories were present in the system and the users generated more than 41,000 consumptions during the test period.

In Figure 12, explicit ratings of applications are compared to the application's implicit rating. Each plotted dot is an application, and the value on the horizontal axis corresponds to this application's average rating in the Android Market. These values have been set by users on a six point scale, from zero to five. The value of the vertical axis corresponds to a computed rating of the application based on

how people have consumed the application. The basic idea of this computation is that installing an application is a good rating. Uninstalling the same application should cancel the good rating out, but give the application some credit as it was obviously interesting in the first place. Placing an application in a list and querying for simillar applications are positive ratings, but less positive than installation. Finally, applications which are considered are given a positive rating as long as they are considered at least every fourth time they turn up in recommendation results. Otherwise, they are given a negative rating. The algorithm for computing these values is presented in Appendix C.

The minimum number of explicit ratings for the applications presented in Figure 12 is 10 and the minimum number of implicit ratings is 20. This way the ratings should be more accurate as there is more data to base the values on. The correlation between the explicit and implicit ratings in Figure 12 is good and in this case, using explicit ratings within the recommender system seems unnecessary. By only using implicit ratings, derived from observing user actions, the user may be liberated from having to put effort in to rating applications explicitly. However, it is not clear how the implicit ratings perform compared to explicit ratings for applications with low ratings.

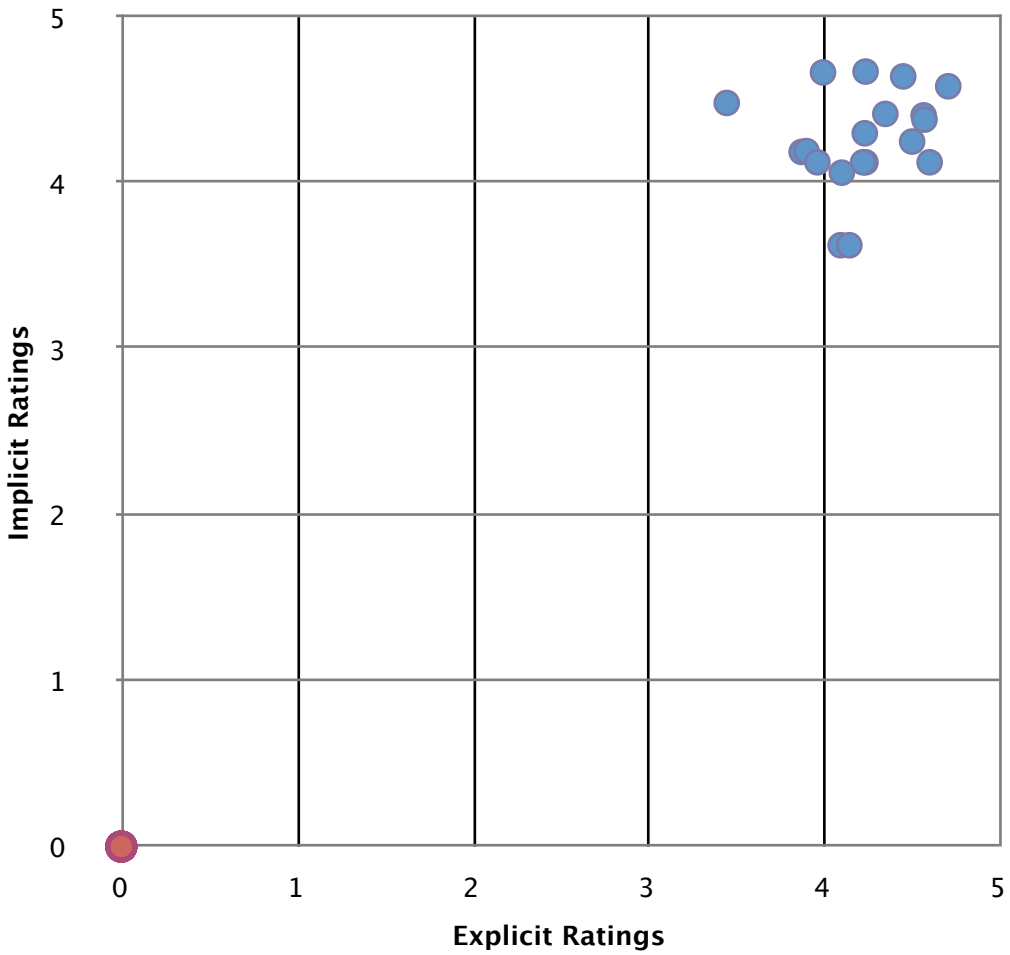


Figure 12. Implicit and Explicit rating correlation for applications with a minimum of 10 implicit ratings and 20 explicit ratings.

In Figure 13, applications are plotted without restrictions on the number of explicit and implicit ratings. From this data, it is apparent that the correlation is

not as good as in the previous case. There is a tendency towards high explicit ratings whereas the implicit ratings are more evenly distributed over the whole rating range. This tendency of high average explicit ratings is analogous to the results of [43], which shows a similar relation between explicit and implicit ratings. Keep in mind that there is very little data for some of the plotted applications. The results shown here are therefore not very reliable, but the correlation of figure 13 is an indication that implicit ratings are promising and the collection of more data along with user studies could prove implicit ratings to be equally good or better than explicit ratings.

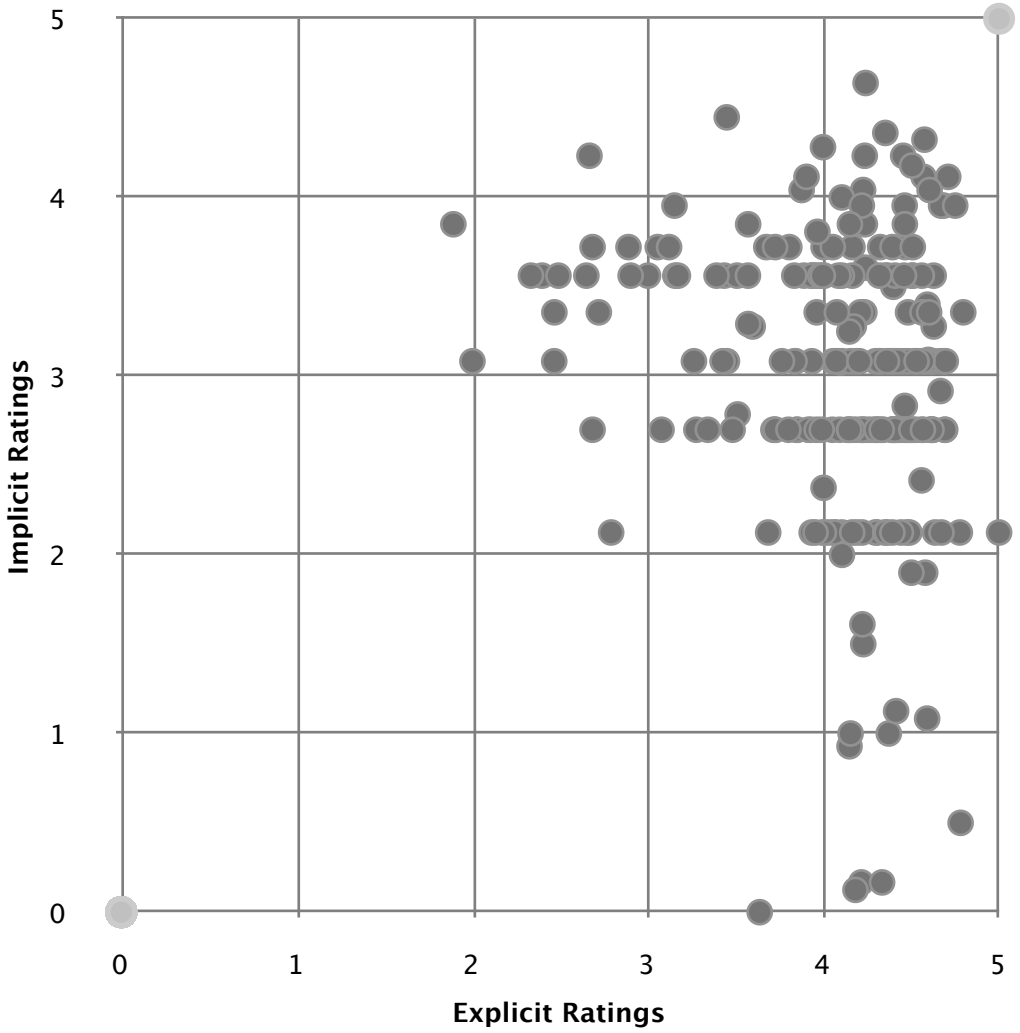


Figure 13. Implicit and Explicit rating correlation without a minimum requirement for the minimum number of ratings.

Figure 14 shows the categories for which applications were run on a user's mobile device during weekends, and before 07.45 and after 17.45 on weekdays. These times corresponds to when a user is usually off work. In Figure 15, consumed application categories are shown for usage during office hours, i.e. between 07.45 and 17.15 on weekdays.

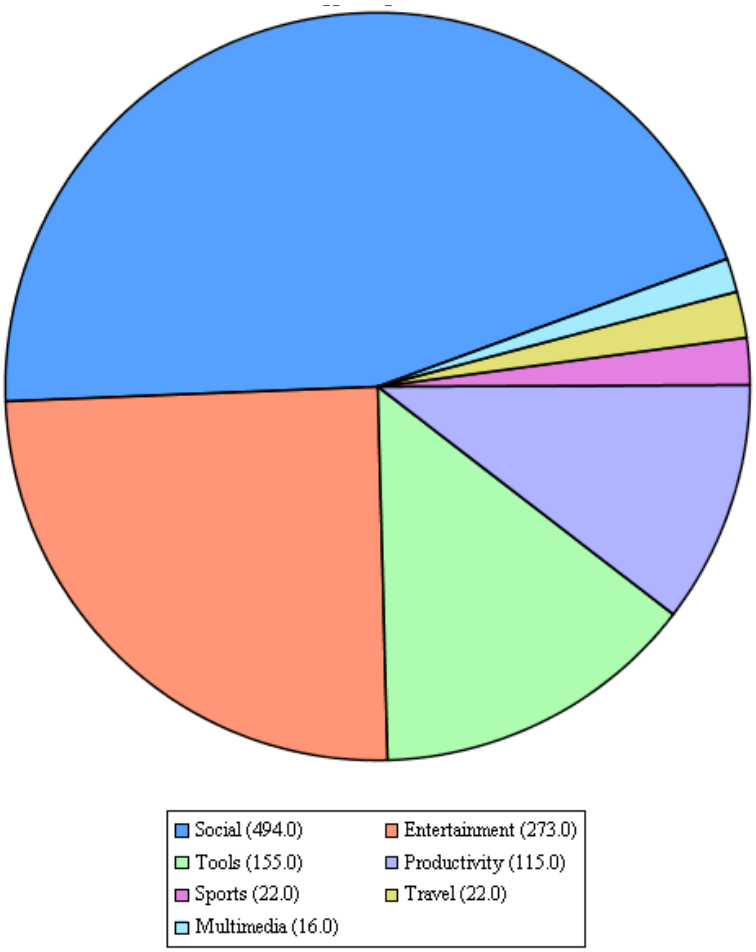


Figure 14. Consumed application categories before 07.45 and after 17.15, and during weekends.

If Figure 14 and Figure 15 are compared, some differences are noted. For instance, during office hours, *tools* is the second most popular category. When users are off work, the second most popular category is *entertainment*. After work, applications of the category *sports* were in fifth place, but during office hours applications of the travel category held this spot. To some extent, this shows that different applications are used at different times during the day. As people tend to live and work at separate locations, it would be safe to assume that different applications are used at different locations as well. This is the expected behavior and it indicates that basing recommendations on context (i.e. time and location) is a method with potential to allow for recommendations to new users who do not yet have a profile or history.

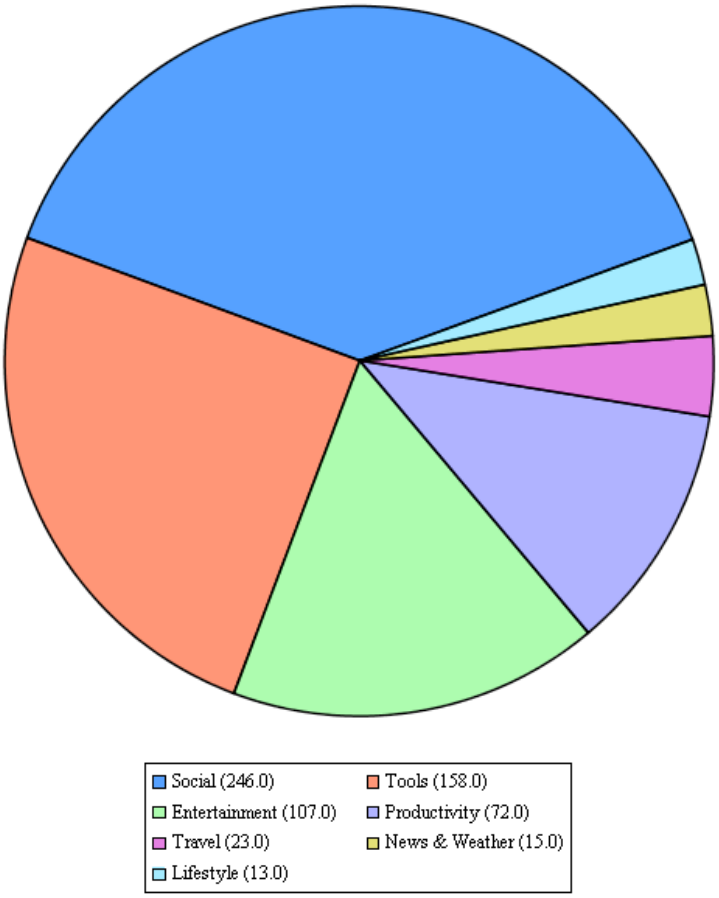


Figure 15. Consumed application categories after 07.45 and before 17.15 during weekdays.

5.5 Methodology

In this section, the methodology of the project is analyzed. The approach is evaluated to guide the methodology of future work.

Several factors influenced the methods applied during the project. Two of these factors were the limited time due to the nature of the project and the absence of data to analyze. The project was to be done in five months and this constraint did not allow for user interviews, user testing, and prototype implementation. Also there was no data or previous results to guide this project from the start.

As a prototype could serve as a platform to perform user tests on and could also generate data, it was a good place to start. Before the prototype could be implemented, the most basic background and previous related work had to be studied and analyzed. In the ideal case, this project should have been preceded by a more thorough investigation involving users and experts to avoid pitfalls and to identify a set of requirements.

Considering the constraints and goals of the project, the methods were suitable. In the event of a continuation of the project, there is a prototype system in place on which user testing and evaluation could be performed.

With the applied methods, the project was however hard to evaluate. The plan for obtaining and analyzing data could have been more elaborate. The qualitative evaluation could also have been improved through more well defined hypotheses and a narrower problem description.

The overall project methodology was satisfactory. However, there could have been more focus on evaluation from the early stages of the project.

6 Conclusions

In helping users to identify and obtain mobile applications of interest, a recommender system was built. The system is a prototype based on a study of previous research and related systems. It shows the potential of application recommendations and identifies the challenges present in the domain, it also provides recommended applications from a user's first launch of the system by utilizing the user's location. Preliminary evaluations were performed.

Due to the great amount of applications for mobile devices being developed and released as of today, the need for recommender systems in this domain is growing. The need is shown by the variety of systems available aiming at solving this problem. Systems such as the one presented in this report show a great potential in solving the problem of aiding end users and a business potential can also be identified. Recommendations of applications would add value to an application store or portal, but there is also a value present in the data generated by the system.

For the implementation of the prototype system, a recommender engine previously used to recommend media items was integrated into the system to allow for personalized recommendations based on users' consumption histories of applications. This integration shows the possibilities of extending current systems to incorporate mobile applications. Recommendations of applications in the prototype system were also based on the user's location by identifying popular applications nearby.

For the limited data available, it was shown that application consumption differs depending on context. This was done by investigating which categories of applications were used at different times of the day. The possibilities of liberating users from having to explicitly rate applications were shown by comparing computed implicit ratings to actual explicit user ratings of applications.

Overall, the goals set out for the project were met and the basic problems and challenges were overcome. However, as the Future Work section shows, there is still a lot to be done in the field.

7 Future Work

There are several challenges awaiting future development of mobile application recommender systems. Some challenges are common to all recommender systems research, and some specifically tied to either context-awareness or applications as items. This section points out these challenges, first in a general manner, and then by looking at aspects specific to the continued development of the prototype system presented in this report.

There are several possible points-of-view to consider when looking at the problem described in this report, there is the one of the end user with a need to identify interesting applications with little effort, without giving up on privacy. There are the researchers with a need of understanding consumers and adapting both algorithms and system interaction to fulfill the user's needs while documenting and evaluating system features. Then there is the business perspective of finding ways to profit from the produced recommendations, which may affect overall system design through impact on research and may affect the utility of the system from the user perspective.

7.1 General

This section highlights the future work of general recommender systems research. Interesting, challenging, promising and important aspects of recommender systems are elaborated on from a long term perspective.

7.1.1 Privacy

Privacy and integrity issues are not limited to the scope of recommender systems, it is an issue in many fields. However, the goal of most recommender systems is to provide the user with *personalized* recommendations, which implies that the system stores some information about each user. Although this personalization is found useful to the users, it may still cause a feeling of concern or discomfort strong enough to turn them away from a service or system. According to [36], privacy risks range from users being afraid that someone else will find out their preferences, to information getting in the hands of stalkers or identity thieves, or even concerns about the information being used in civil litigation or for government purposes. As [36] points out, these concerns and actual privacy risks are amplified by new devices giving access to the user's contact list, precise physical location and other sensory data.

Developing and utilizing techniques which allow for personalized recommendations and minimal privacy risks should be address by future research, not only to build user trust, but also for their safety.

7.1.2 Context

The necessity of context-awareness in recommender systems depends greatly on the type of items to be recommended. Some items, like news articles, are more relevant during a certain period in time, whereas some items, like subway map applications, are more relevant at a certain location. In such cases, context-awareness in a recommender system would contribute to better recommendations in terms of relevance, but may not be absolutely essential.

However, some recommender systems such as point-of-interest and travel activity recommender systems need to be context-aware as the items to be recommended are greatly made up of, or extensively connected to, context. As [33] points out, an outdoor flea market may be a good recommendation on a sunny day, but a bad recommendation on a rainy day. This characteristic of items, the switch from a good to a bad prediction depending on context is what causes context-awareness to be necessary in such recommender systems, as irrelevant recommendations are a source of decreased user trust and satisfaction [35].

The most apparent benefits of context-awareness in recommender systems are increased accuracy and user trust. The removal of non relevant or inappropriate recommendations is the main source of these benefits, but there is also the potential of extended serendipity as [33] argues. By basing recommendations not only on the content of items, or a neighborhood of users, items which are truly new and unexpected may be recommended based on the context. The prototype system in this article captured this characteristic of context dependent items to reduce the effect of the *new user problem*. This solves one problem to some extent, but introduces the *new context problem,* which needs to be addressed in future research.

The *new context problem* is the problem of basing recommendations on a context which has yet to be seen by the system. If there is no item consumption for this new context, the system is not able to produce recommendations, in analogy with the new user problem for collaborative filtering.

Apart from the new context problem, future research is facing the challenge of identifying relevant contextual information and interpreting it to make it useful in a recommendation process. This is highly dependent on the type of items to be recommended and the available contextual data. For instance, with mobile devices having various sensors and GPS-functionality the contextual data available is growing, but it may not all be useful, so identifying which information to base recommendations on will be a challenge. Also, some items may be more bound to context than other, news article recommendations seem to be more context dependent than friend recommendations in a social network, but this may not be the case, and understanding this is important for future work. This understanding of context, items and recommendations would require data, and the prototype of this report may be able to come up with that kind of data.

7.1.3 Observed User Actions

As stated previously, the accuracy metric of recommender systems does not tell the whole story, high accuracy is however not a negative feature. In order to take

full advantage of user feedback and implicit user actions to produce highly accurate and relevant recommendations, the user's actions (implicit ratings) will need to be understood. The challenge is that actions are often domain dependent and different actions are observable in different domains. For instance, there are major differences in recommending mobile applications in a browser and on a mobile device. On a device it is possible to observe how frequently a mobile application is used and when they are installed and uninstalled, which is not possible in a broeser. For common domains (e.g. friend recommendations on the internet), a set of best practicies based on observations could be formulated.

7.1.4 Human Interaction and System Utility

A contributing factor to user satisfaction is the recommender systems ability to explain the recommendations the user. This is a belief extensively spread through the research community. There is however not a clear or unified way of incorporating explanations in a consistent way across recommender systems. Another positive aspect of explanations is their ability to educate the user in the system. Explanations could teach a user what the consequences of her actions are. Simply displaying an explanation for the sake of having explanations is however not providing to this educational aspect as misleading and not accurate explanations could have the opposite effect. Making sure how users perceive and react to explanations will have to be further investigated.

7.1.5 Business Perspective

The increased use and development of recommender systems has its roots in e-commerce, for which recommender systems is an important part in increasing sales. Items recommended to users, which have accuracy above random are naturally more likely to be clicked, and in the end bought. This may be motivational enough for incorporating recommender systems, but there could be other business models for generating revenue, not only to e-commerce sites and portals.

For instance, Last.fm [14] - a music recommender site - does not profit from selling CDs or even music. In a way, they sell their users, not only by having advertisement on the site. They also provide special accounts for record labels and artists, which include purchasable features such as Last.fm-radio time, directed at users with a specified taste. [40]

Identifying such business models could drive the field forward by bringing money in to research and development. Recommender systems have the potential to be useful to users and companies alike.

7.2 Prototype System

Future work specifically tied to the continuation of this thesis project is presented in this section. The long term work on the system is not limited to

recommendations research, but also involves other fields, which is reflected in the topics below.

7.2.1 Portability versus Privacy

As each user of the prototype system presented in this report is identified solely through a randomly generated string only linked to the user through the device, user profiles are anonymous. This is a good way to address some privacy concerns [36]. However, this comes at the price of making the system non-portable, i.e. the data of the system could not be used by any other client than the device on which the user first launched the system.

One way to solve this could be to let the user choose how and if she should be anonymous, depending on the intention of future use.

7.2.2 Understanding and Interpreting User Actions

The prototype system currently observes and collects more user actions than the ones being used and interpreted to produce recommendations. In order to make use of all user actions they need to be understood. It is not clear as of today what a *mark* consumption type actually means, i.e. is it a bad or a good rating. Is an uninstall action always a bad rating or did the user actually like the application? A free but limited version of an application may be liked by the user, to the extent that the user buys the full version of the application and uninstalls the free version.

Such scenarios make interpretation of implicit ratings difficult, but using the data collected by the prototype system presented in this report, a study could be performed to obtain a better understanding of user actions.

7.2.3 Context

The system currently stores location and time data along with consumptions. The contextual data could be extended to include more attributes. A study would then have to be performed to analyze how the new contextual attributes affect application usage, before they could be put to use in improving recommendations.

7.2.4 Scalability and Performance

The scalability of the system has not been evaluated. No performance tests or stress tests have been applied during the project. This is a concern, which must be dealt with during the future work of the system. The ideal case would be to have millions of users producing a lot of consumption data, as this would allow for the best possible recommendations.

8 References

[1] Adam Mustansar Ali Ghazanfar and Adam Prugel-Bennett, “A Scalable Accurate Hybrid Recommender System”, in Third International Conference on Knowledge Discovery and Data Mining, 2010, pp. 94-98.

[2] Gediminas Adomavicius and Alexander Tuzhilin, “Towards the Next Generation of Recommender Systems: A Survey of the State-of-the-Art and Possible Extensions”, IEEE Transactions on Knowledge and Data Engineering, Vol. 17, No. 6, 2005, pp. 734-749.

[3] Andrew I. Schein, Alexandrin Popescul, Lyle H. Ungar and David M. Pennock, “Methods and Matrics for Cold-Start Recommendations”, ACM, 2002.

[4] Francesco Ricci, “Mobile Recommender Systems”, 2010.

[5] Wolfgang Woerndl, Christian Schueller and Rolf Wojtech, “A Hybrid Recommender System for Context-aware Recommendations of Mobile Applications”, IEEE, 2007.

[6] Wolfgang Woerndl and Georg Groh, “Utilizing Physical and Social context to improve Recommender Systems”, IEEE, 2007.

[7] Kolja Hegelich & Dietman Jannach, “Effectiveness of different recommender algorithms in the Mobile Internet: A case study”, AAAI, 2009.

[8] Mark Dunlop & Stephen Brewster, “The challenges of Mobile Devices for HCI”, Personal and Ubiquitous Computing, 2002.

[9] Maryam Hosseini-Pozveh, Mohamadali Nematbaksh and Naser Movahhedinia, “A multidimensional approach for context-aware recommendation in mobile commerce”, in International Journal of Computer Science and Information Security, 2009.

[10] Adomavicious, G., Sankaranarayanan, R., Sen, S., and Tuxhilin, A., ”Incorporating contextual information in recommender systems using a multidimensional approach”, ACM Trans. Inf. Syst., 23(1):103-145, 2005.

[11] Karen Church, Barry Smith, and Mark Keane, “Evaluating Interfaces for intelligent Mobile Search”, 2006.

[12] Quang Nhat Nguyen, Francesco Ricci, and Dario Cavada, “Critique-based Recommendation for Mobile Users: GUI Design and Evaluation”, 3rd International Workshop on ‘HCI in Mobile Guides’, 2004.

[13] Cena, F., Console, L., Gena, C., Goy, A., Levi, G., Modeo, S. and Torre, I. “Integrating heterogeneous adaptation techniques to build a flexible and usable mobile tourist guide”, AI Commun, 19(4):369-384, 2006.

[14] http://www.last.fm (29 November, 2010)

[15] Upendra Shardanand, and Pattie Maes, “Social Information Filtering: Algorithms for Automating ‘Word of Mouth’”, ACM, XXXX.

[16] Vincent W. Zheng, Bin Cao, Yu Zheng, Xing Xie, and Qiang Yang, "Collaborative Filtering Meets Mobile Recommendation: A User-centered Approach", AAAI, 2010.

[17] Jonathan L. Herlocker, Joseph A. Konstan, and John Riedl, "Explaining collaborative filtering recommendations". In Proceedings of the 2000 ACM conference on Computer supported cooperative work (CSCW '00), ACM

[18] Robert Bell, Yehunda Koren, and Chris Volinsky, "Modeling relationships at multiple scales to improve accuracy of large recommender systems", Proceedings of the 13th ACM SIGKDD international conference on Knowledge discovery and data mining, pp. 95-104, 2007.

[19] Shean M. McNee, John Riedl, and Joseph A. Konstan, "Being accurate is not enough: how accuracy metrics have hurt recommender systems", Conference on Human Factors in Computing Systems, pp. 1097-1101, 2006.

[20] Hyung Jun Ahn, "A new similarity measure for collaborative filtering to alleviate the new user cold-starting problem", Information Sciences, 178(1), pp. 37-51, 2008.

[21] David M. Nichols, "Implicit Rating and Filtering", Proceedings of the fifth DELOS Workshop on Filtering and Collaborative Filtering, 1997.

[22] Cai-Nicholas Ziegler, Shean M. McNee, Joseph A. Konstan, and Georg Lausen, "Improving Recommendation Lists Through Topic Diversion", W3, ACM, 2005.

[23] http://googleblog.blogspot.com/2009/12/personalized-search-for-everyone.html (July, 2010)

[24] http://appsfire.com (17 July, 2010)

[25] http://www.appspace.com (17 July, 2010)

[26] http://appsaurus.com (19 July, 2010)

[27] http://developer.android.com/guide/basics/what-is-android.html (20 July, 2010)

[28] Guiran Chang, Chunguang Tan, Guanhua Li, and Chuan Zhu. "Developing Applications on the Android Platform", Springer, 2010.

[29] http://www.openhandsetalliance.com/ (23 July, 2010)

[30] Yuli Vasiliev, "Beginning Database-Driven Application Development in Java EE: Using GlassFish", Apress, 2008

[31] http://developer.android.com/sdk/installing.html (1 September, 2010)

[32] Tao Zhou, Zoltan Kuscik, Jian-Guo Liu, Matus Medo, Joseph Ruschton Wakeling, and Yi-Cheng Zhang, "Solving the apparent diversity-accuracy dilemma of recommender systems", 2010

[33] Girardello, A. and Michahelles, F. 2010. AppAware: Which Mobile Applications Are Hot? Proceedings of the 12th International conference on Human Computer Interaction with mobile Devices and Services

[34] Böhmer, M. and Bauer, G. Exploring the Design Space of Context-Aware Recommender Systems that Suggest Mobile Applications. CARS-2010

[35] Said, A. Identifying and Utilizing Contextual Data in Hybrid Recommender Systems. RecSys 2010

[36] Faith Cranor, L. ´I Didn't Buy it for Myself´ Privacy and Ecommerce Personalization. WPES 2003

[37] http://code.google.com/p/google-gson/ (1 November, 2010)

[38] http://www.json.org/ (1 November, 2010)

[39] Shahan, M. Personalized TV with Recommendations – Integrating Social Networks. Master thesis, 2008
[40] http://musicmanager.last.fm/ (8 November, 2010)
[41] http://code.google.com/p/android-market-api/ (11 November, 2010)
[42] http://www.androlib.com/ (16 November, 2010)
[43] Girardello, A. and Michahelles, F. "Explicit and Implicit Ratings for Mobile Applications". ETH Zurich. 2010.

9 Appendix A

Media Metadata	Application Metadata
Title	Title
Description	Description
Genres	Category
Keywords	Version
Contributors	Creator
Languages	Install size
Release Date	
Awards	
Length (time)	
Organization	

Table 3. Common attributes / metadata comparison between media (movie) and applications (Android).

Media Metadata	Application Metadata
Forrest Gump	Google Maps
Forrest Gump, while not intelligent, has accidentally been present at many historic moments, but his true love, Jenny, eludes him.	**New features: * See ratings for aspects about a place such as service or atmosphere**
Drama, Romance	Travel
Vietnam, 1950s, 1970s, Bus, Destiny, Shrimp, Fishing, Table tennis, Simple man, etc	4.3.0
Robert Zemeckis, Winston Groom, Tom Hanks, Robin Wright	Google Inc.
English	20kb
14 october 1994	
Oscar, Oscar, Oscar, Oscar	
142 min	
Paramount	

Table 4. Example of metadata values for a movie and application respectively.

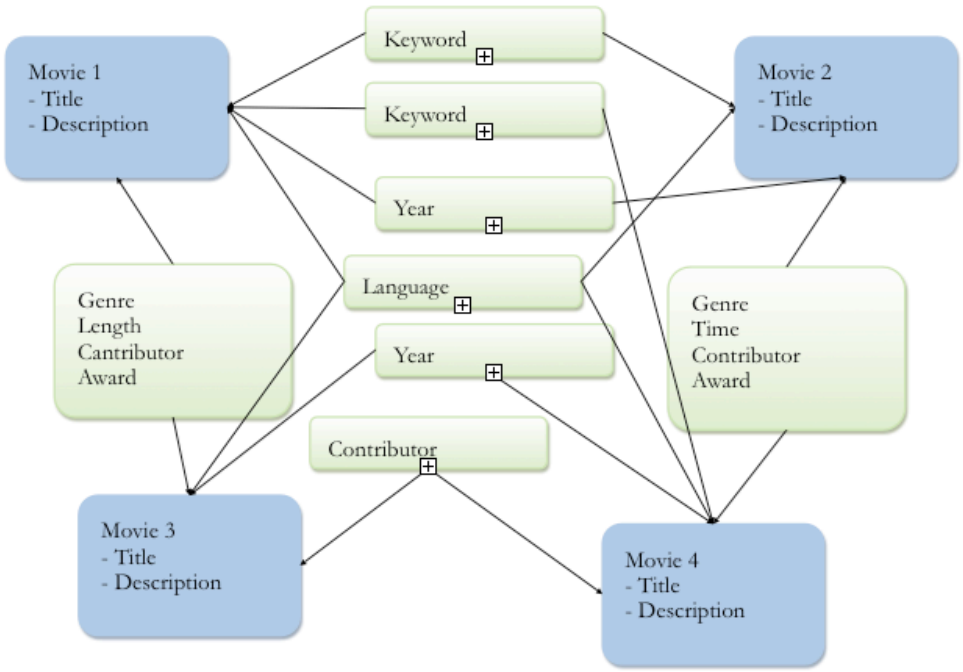


Figure 16. Conceptual semantics of movie metadata. Attributes are shared among m movies.

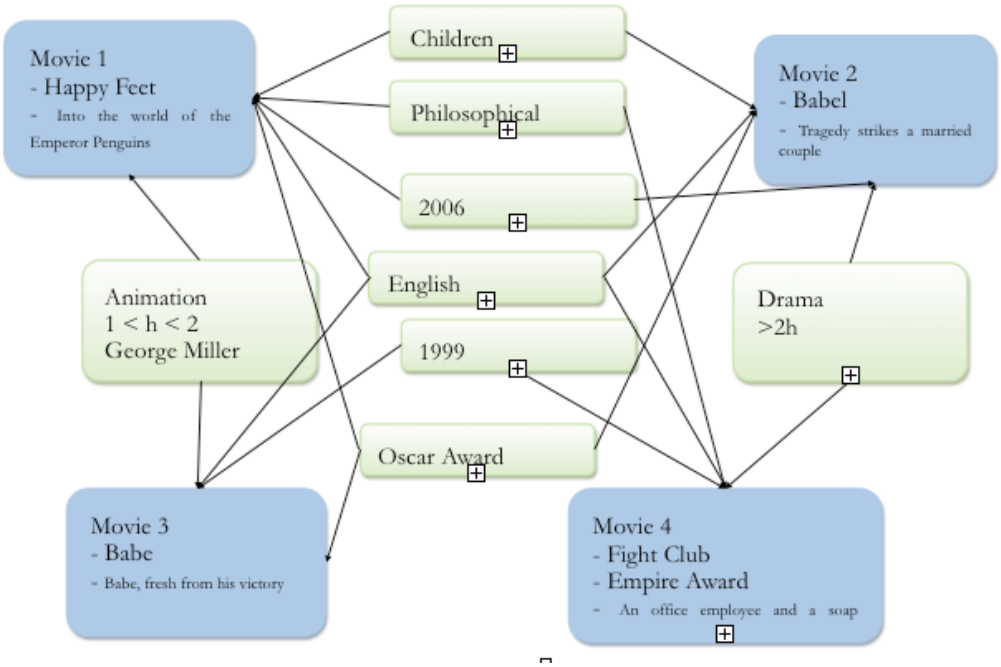


Figure 17. Example semantics of movie metadata. Attributes are shared among movies.

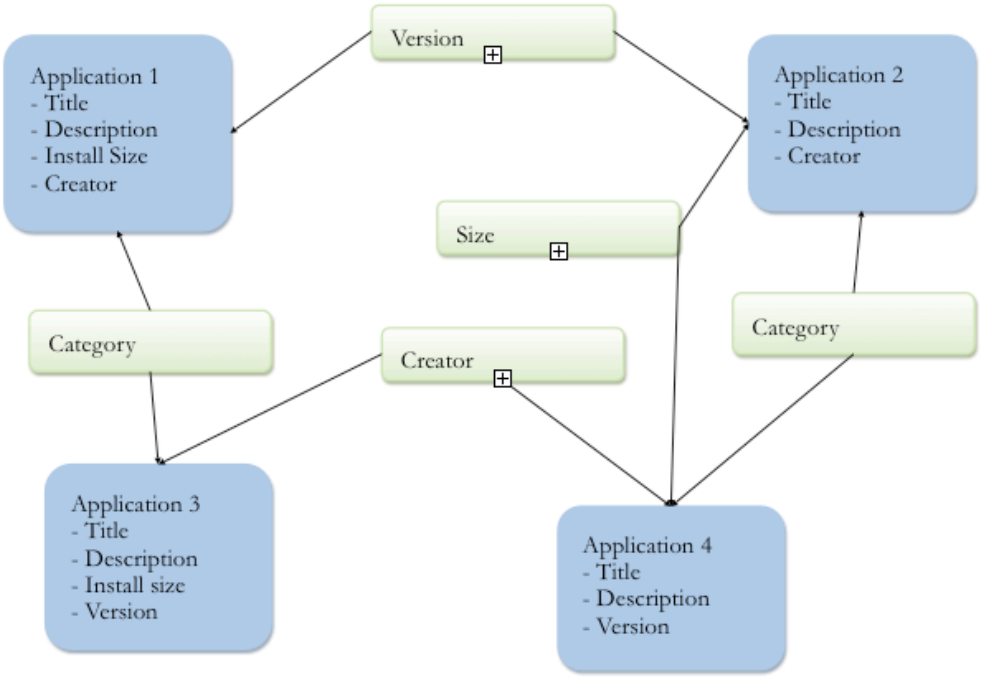


Figure 18. Example semantics of movie metadata. Attributes are shared among movies.

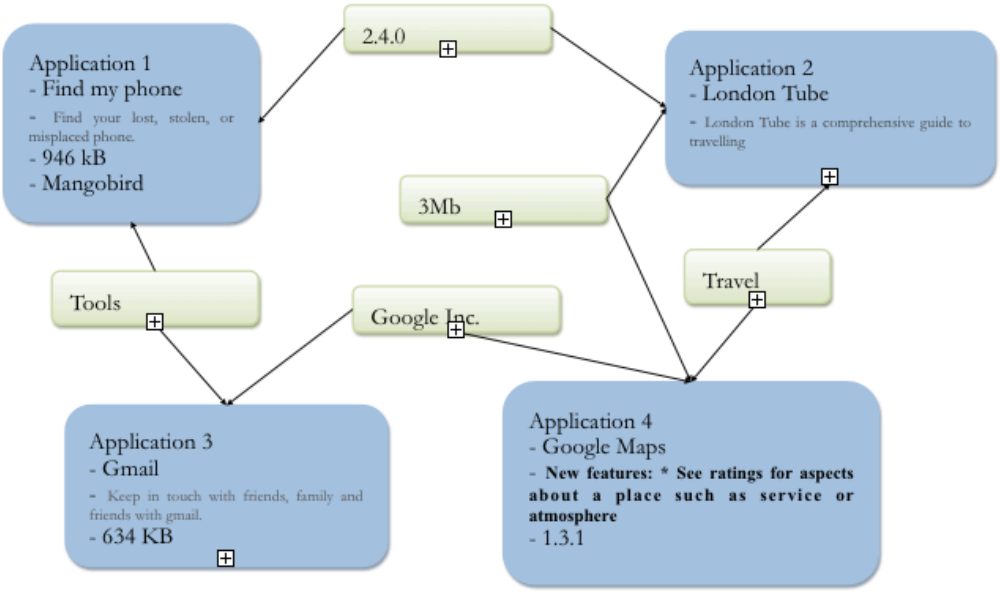


Figure 19. The conceptual semantics of application metadata. Attributes are not as meaningful; they are not shared to the same extent.

10 Appendix B

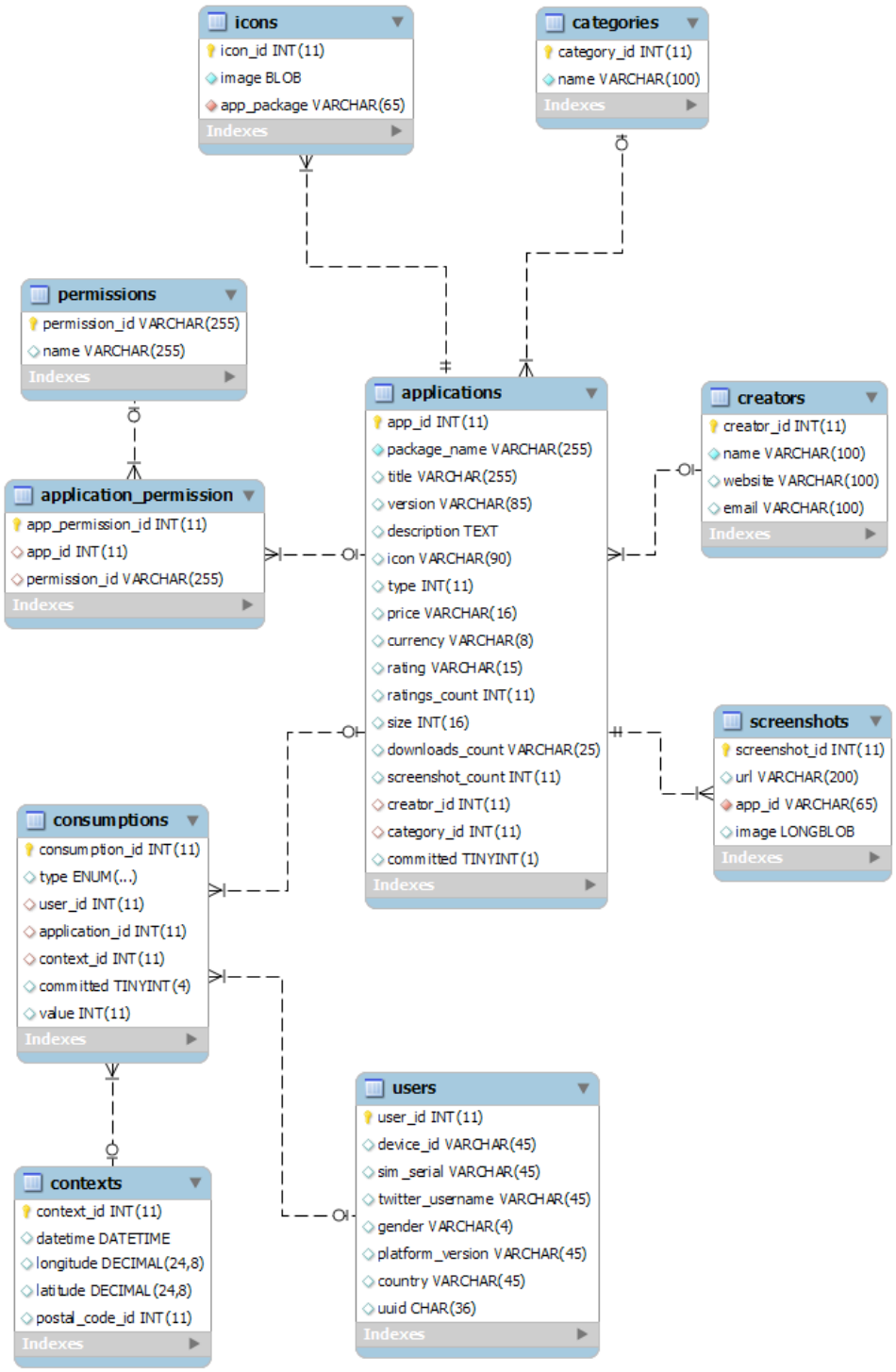


Figure 20. Database model for the server of the system.

11 Appendix C

Algorithm for computing the implicit rating of an application:

```
number_of_ratings = 0
Implicit_rating_of_app = 0
For each consumption of app
{
          If(consumption_type = Save)
             Implicit_rating_of_app =
          Implicit_rating_of_app + 100
             number_of_ratings = number_of_ratings + 1
          If(consumption type = Delete)
             Implicit_rating_of_app =
          Implicit_rating_of_app -90
             number_of_ratings = number_of_ratings + 1
          If(consumption type = Mark)
             Implicit_rating_of_app =
          Implicit_rating_of_app + 10
             number_of_ratings = number_of_ratings + 1
          If(consumption type = Query)
             Implicit_rating_of_app =
          Implicit_rating_of_app + 20
             number_of_ratings = number_of_ratings + 1
          If(Count(Considers)*2 > Count(Glimpse)
             Implicit_rating_of_app =
          Implicit_rating_of_app + 30
             number_of_ratings = number_of_ratings + 1
          If(Count(Considers)*4 > Count(Glimpse)
             Implicit_rating_of_app =
          Implicit_rating_of_app + 5
             number_of_ratings = number_of_ratings + 1
          If(Count(Considers)*4 <= Count(Glimpse)
             Implicit_rating_of_app =
          Implicit_rating_of_app - 30
             number_of_ratings = number_of_ratings + 1
}

return implicit_rating_of_app / number_of_ratings
```