FREE UNIVERSITY OF BOLZANO
FACULTY OF COMPUTER SCIENCE

Real-Time Context-Aware Recommendations for Mobile Users

Thesis submitted for the Bachelor of Science in Applied Computer Science

Author:
Stefan Peer

Supervisor:
Prof. Dr. Francesco Ricci

Co-Supervisor:
Linas Baltrunas

Academic Year 2009/2010

Dedicated to my family

Acknowledgements

I would like to thank everyone who helped me realizing this work and supported me during the last three years of study.

- My university tutor Prof. Dr. Francesco Ricci who proposed this thesis to me and led me in the right direction.
- My co-supervisor Linas Baltrunas from the DIS (Database and Information Systems) research group who supported me in all project tasks.
- The Sinfonet kGmbH, which provided us their touristical data.
- My parents, Martha and Arnold Peer, who supported me all the time and allowed me to pursue this study.

Lastly I would like to give a special thank to Prof. Ricci, who provided me his MacBook Pro for about six months. Without a Apple notebook I would not have been able to implement the iPhone application.

Thank you all! Stefan

Abstract

The research project of this thesis comprises the design, implementation and test of a mobile user interface for a real-time, context-aware recommender system. The core recommendation techniques have been developed by the DIS group at the university of Bolzano within the ReRex research project. My task was to develop an iPhone application, which is able to connect to the ReRex recommendation service and deliver recommendations and their real-time updates to the users. We used a client-server architecture for implementing the system, i.e., the iPhone client that communicates with an XML webservice on the server. The two components are based on different software infrastructures: while the server component is based on Java and runs in an Apache Tomcat servlet container, the iPhone application is written in Objective C, using the Cocoa Touch framework.

Generally ReRex technology can be used for providing recommendations of every kind. In this project we collaborated with suedtirol.info, a South-Tyrolean tourism portal provided by *Sinfonet KGmbH* and *Südtirol Marketing Gesellschaft KGmbH*. They provided us a webservice from which we could access their data set of touristical points of interest. Using this data, we set up, configured and trained ReRex recommendation techniques. The ultimate goal of this thesis project was to understand whether the *context* can make the city guide more useful, i.e., if the availability of contextual information can influence the recommendation generation in a positive way, i.e., the users are more satisfied with the recommendations computed by the system taking into account the contextual factors.

A major challenge during the project has been represented by the user interface design. Designing a user interface for a mobile system requires to take into consideration various issues related to hardware limitation and the specific - mobile - context of usage. The interface should contain all intended functions, should be understandable, easy to use and as user friendly as possible. A usability test, conducted at the end of the development, showed that our application does not have any major usability problem and that the context management characteristic of our system improves the users' acceptance.

Abstract

Das Forschungsprojekt dieser Diplomarbeit besteht aus dem Design, der Implementierung und dem Test von einer Benutzeroberfläche für ein kontextbezogenes Empfehlungssystem für mobile Geräte, welches Aktualisierungen in Echtzeit liefert. Das Empfehlungssystem selbst wurde von der DIS Gruppe an der Freien Universität Bozen innerhalb des ReRex Forschungsprojekts entwickelt. Meine Aufgabe war es, eine iPhone Applikation zu entwickeln, welche sich mit dem ReRex Empfehlungsservice verbindet und dessen Benutzern Empfehlungen und Echtzeitupdates zur Verfügung stellt. Dabei kam eine Client-Server Architektur zum Einsatz: Das iPhone, der Client, kommuniziert mithilfe eines XML Webservice, welcher auf dem Server läuft. Beide Komponenten basieren auf verschiedenen Architekturen: die Serverkomponente wurde in Java programmiert und läuft im Apache Tomcat Servlet Container. Für die iPhone Applikation, welche in Objective C geschrieben ist, wurde das Cocoa Touch Framework verwendet.
Grundsätzlich kann die ReRex Technologie für jede Art von Empfehlungen verwendet werden. In diesem Projekt haben wir mit suedtirol.info, einem Südtiroler Toursimusportal von der *Sinfonet KGmbH* und der *Südtirol Marketing Gesellschaft KGmbH*, zusammengearbeitet. Diese haben uns einen Webservice zur Verfügung gestellt, von welchem wir touristische Sehenswürdigkeiten beziehen konnten. Mit den gewonnenen Daten konnten wir die ReRex Empfehlungstechnologien einrichten, konfigurieren und trainieren. Das Endziel dieses Projekts war es, zu untersuchen, ob der Stadtführer durch die Verwendung von kontextbezogenen Daten nützlicher und benutzerfreundlicher wird. Wir prüften, ob zusätzliche kontextabhängige Informationen die Erzeugung der Empfehlungen in positiver Weise beeinflussen können, d.h. ob Benutzer mit einem kontextbewussten Empfehlungssystem zufriedener sind, als mit einem System welches Empfehlungen nur auf Basis von generellen Benutzervorzügen abgibt.
Ein wichtiger Aspekt dieses Projekts war das Design der Benutzeroberfläche. Mobile Systeme erfordern eine besondere Art und Weise, Bedienoberflächen zu kreieren: einerseits ist die Hardware limitiert, und andererseits muss das Programm dem Mobilitätsaspekt angepasst werden. Alle gewünschten Funktionen sollten enthalten sein, aber trotzdem sollte die Applikation verständlich, leicht benutzbar und so benutzerfreundlich wie möglich sein. Nachdem die Entwicklung abgschlossen war, wurde ein Usability-Test durchgeführt. Dieser zeigte, dass die iPhone Applikation keine größeren Probleme aufweist, und dass das Kontext-Management des Systems von den Benutzern positiv empfunden und akzeptiert wurde.

Riassunto

Il progetto di ricerca di questa tesi comprende la progettazione, la realizzazione e la valutazione di un interfaccia grafica di un sistema di "recommendation" di punti di interesse in una citta, dove i suggerimenti sono adattati al contesto dell'utente in tempo reale. Le tecniche di "recommendation" sono state sviluppate dal gruppo DIS presso l'Università di Bolzano nell'ambito del progetto di ricerca ReRex. Il mio compito era quello di sviluppare una applicazione per iPhone, in grado di connettersi al servizio di "recommendation" ReRex e fornire suggerimenti in tempo reale agli utenti. Abbiamo utilizzato una architettura client-server per la realizzazione del sistema. Il client, cioè l'iPhone, comunica tramite un XML webservice col server. I due componenti sono basati su differenti infrastrutture: mentre la componente server è basata su Java e viene eseguita nel Apache Tomcat Servlet Container, l'applicazione per l'iPhone è scritta in Objective C, utilizzando il Cocoa Touch framework.
La tecnologia ReRex può essere utilizzata per fornire proposte di ogni genere, ma in questo progetto ci siamo focalizzati sul dominio del turismo. Abbiamo collaborato con suedtirol.info, un portale turistico sudtirolese della *Sinfonet KGmbH* e *Südtirol Marketing Gesellschaft KGmbH*, il quale ci ha fornito un webservice dal quale abbiamo potuto accedere ai loro dati sui punti di interesse turistici.
Usando questi dati, abbiamo installato, configurato e sviluppato le tecniche di "raccomandazione" ReRex. L'obbiettivo principale di questo progetto era comprendere se le informazioni sul contesto del suggerimento, come per esempio la composizione del gruppo dei visitatori o il loro mezzo di trasporto, potesse migliorare la qualità del servizio informativo fornito dal sistema.
Una delle difficoltà maggiori affrontate nel corso del progetto è stato il disegno dell'interfaccia grafica. La progettazione dell'interfaccia per un sistema mobile richiede l'analisi di vari aspetti. In particolare, si deve tener conto delle limitazione dell'hardware (schermo e capacità di calcolo) e del contesto di utilizzo mobile. Inoltre l'interfaccia doveva supportare tutte le funzioni previste, doveva essere comprensibile, facile da usare e più chiara possibile. Un test di usabilità, condotto alla fine dello sviluppo, ha dimostrato che la nostra applicazione non presenta problemi di usabilità e che le informazioni contestuali usate dal sistema di "raccomandazione" rendono l'applicazione più efficace nell'aiutare gli utenti nella ricerca di attrazioni turistiche di loro interesse.

Contents

1. Introduction

1.1. Context of the research

Nowadays people are overflowed with information from everywhere. Mass media and especially the Internet provides plenty of data and it's hard to extract only what is really needed. Because of this overabundance of information it may get hard for people understanding an issue and making decisions [1]. An approach for partly solving this problem are search engines. By specifying keywords a user is able to search in the World Wide Web, but the problem there is that, millions of pages will be provided and it's not always easy to find the needed, reliable information. Nowadays search engines are very advanced, so that they are capable to analyse users behaviour in the web and exploit this information in the ranking of the search results. So users should easier find what they need. The same occurs with online shops, as for example amazon.com. When users search for a product they are advised about what other people, which have bought that product, have purchased additionally. It's a personalised information that might be interesting for them and could help to continue shopping. This information service has been shown to increase user satisfaction, because it detects customers needs and might stick them to a business. These services are designed for helping users to deal with information overload, providing filtered and personalised information about products or services that the user may find interesting and useful. They are called recommender systems [2] and they are now very popular in many online shops (amazon.com, yahoo.com, etc.) and media web sites (iTunes store, youtube, last.fm, etc.).

In our research, we wanted to expand a recommender system with two important aspects:

- *Context awareness*, i.e., the ability to consider contextual factors in the generation of recommendations. For example: a person asks a friend for a walking path recommendation. He knows her personal preferences, but this might not be enough to give a suggestion. An important aspect might also be, with whom the person wants to go to hike: what is the companion? Is it the family, with children? The colleagues? The boy/girlfriend? The recommendation will therefore depend on the context.

- *Real-Time* refers to the ability of the system to react to situation changes and inform the user about them. Consider the previous example: the person decided to make a 7 hour trip to a mountain with her colleagues. The day before the trip weather forecasts unexpectedly predict heavy raining for the next day. A real-time system is able to react to the context change (weather), to notify the user about the situation and eventually to provide alternatives.

1.2. Problems addressed

Our research focused on the development of a mobile application for providing context-aware and real-time recommendations to tourists in Bolzano. After the user has specified her profile the system should generate a list of items, called suggestions or recommendations, that are adapted to the current context of interaction. In addition the system should be able to react on real-time changes, i.e., to changes of the user preferences or contextual conditions.

The major issue during the development of this application was the design of a user interface as usable as possible. Mobile applications require a special attention to usability and they are designed prioritizing criteria that are different from those normally considered in desktop applications. For instance, in our particular application was very important to clarify when and how often to notify the user about contextual changes (*real-time*). Too many notifications might annoy the user and it may result that she will ignore them forever. Too few may lead to malfunctionality of the *real-time* feature. There could be several types of changes in the recommendations produced by context changes, but in this work we focussed on two of them: *add* and *delete*.

Finally, we wanted to test if context can be exploited for generating better recommendations, i.e., more accurate and more easily accepted by the user. We expected this positive effect, since context is providing additional and useful information for the recommender system [3].

At the end of the system design and implementation we performed a user study for evaluating the usability of our mobile application. The long term objective of our project is to improve this application so that it can be offered through the App Store.

1.3. Approach and Solution

To compute the recommendations we have used ReRex, a context-aware recommender system, developed by the DIS group at the university of Bolzano. In this project we focussed on tourism, because in South-tyrol it's a very important source of revenue and it is easy to obtain information about the items to recommend, i.e., points of interest. In fact, data was provided by suedtirol.info, which has a huge database of locations, distributed over the whole province. They comprise several categories such as museums, nature parks, walking paths, castles, etc. For our research purposes we decided to use only POIs in Bolzano, but the database is still growing.

A very important issue regarding recommendations is how many POIs to show to the user. We decided that the system must provide around 6 recommendation items, to not overload the user with too many information and decision possibilities. If a user is not satisfied with these 6 recommendations, we offer the option to browse through all the items of a given category.

The system should support tourists, i.e., people that are new to Bolzano, but also local people to plan their free time, holiday or business activity. One might think that this can also be reached by going on a tourist office. This is only partially true since

a recommender system provides points of interest which are personalised to the user's wishes. A tourist office cannot know each single tourist and her preferences. In additon it doesn't know much about the contextual conditions occurring during the visit, i.e., how will be the weather, if they will move by foot or by bus, etc.

1.3.1. Why mobile?

RS are normally accessed from a simple desktop application or website at home, but we decided to create a mobile application because of severeal reasons:

1. People have their mobile devices always with them, so they will be immediately notfied of real-time changes. It's useless if the notification arrives at the home PC while one is at a trip.

2. Context is also location dependent: in a mobile application there is the possibility to access GPS devices and accelerometers. While the user changes location, also context may change and the user will be notified immediately. At the PC at home, location will not change.

3. People are unpredictable: they might decide from one moment to another to undertake something, where they might need a recommendation. If they are not at home, but already on a trip they can ask their mobile device.

These factors are very crucial, but one has also to look at the other side of the coin. A very important issue regarding mobility is the user interface. A mobile device will not have a large 19" screen, but about 3-4". There is not the same space as on a desktop PC, so the user interface has to be cautiously adapted to the task and the context. Additionally there shouldn't be too many controllers and functions on a single view, otherwise the user gets distracted and feels lost in the application.

1.3.2. Why iPhone?

"The iPhone was a game changer in the mobile ecosystem." [5]
We decided to develop an iPhone application, because it's one of the most popular phones and it is a very fast growing mobile platform. Statistics of Gartner Inc., the worlds leading information technology research and advisory company, have shown that in the first quarter 2009 market share of the iPhone OS was 10.5%. 2010 it raised to 15.4% [6]. More than a billion mobile applications have been sold for these device in under a year. The functionality of Apples iPhone SDK allowed us to integrate all the desired features, such as real-time notifications. Another advantage of the iPhone is its App Store, the only official download platform for iPhone applications. It's an easy way to distribute an application.

1.3.3. Why an application, not a website?

On iPhone there exist two possibilities in providing services to user: a website, optimized for mobile devices or a native mobile application. Both have their pro and cons but finally

we decided to make an application because of the following reasons [5]:

- They offer a best-in-class user experience, offering a rich design and tapping into device features.
- Better access to the devices peripherical devices such as GPS, accelerometer, etc.
- An application can run in background and perform background tasks, i.e., notify the server about contextual changes.
- It can also be used offline.
- It's a source of revenue, since it can be sold easily on the App Store platform.

Surely an iPhone application has also some disadvantages. The major is that it is not platform independent. A mobile website can be accessed by any device which has a browser. It will not be displayed always in the same way, but generally it can be accessed. A native application is written for a determined platform such as iOS, Android, J2ME, Symbian, etc. and cannot be used for another one. Nevertheless because of the features described above we decided to develop a native application.

1.4. Results of the work

When we had the idea for this application we were not familiar with the Cocoa Touch framework. Step by step we had to learn the details by ourself. To offer the desired functionality we used many different technologies that the iOS platform provides, i.e., networking, XML parsing, GPS device, background tasks, local- and push notifications, mapkit, navigation controllers, memory management, etc. It was all new to us on this platform, therefore we often got stuck and had to search new solutions for implementation problems. Finally our work resulted in an iPhone application which provides real-time context-aware point of interest recommendations to mobile users. Users can specify profile and contextual conditions and based on this two factors the system generates point of interest suggestions. Users can browse through them, watch them on a map and put them in a wishlist, i.e., a list of favourite POIs. After having visited a POI they can also give a rating to it.

For verifying the usability of our application we conducted a user study. We tested whether the context can influence the user decisions in a positive way and lead to more appropriate recommendations. To test our hypothesis we used a Computer System Usability Questionnaire [4], which the subjects had to fill out after trying out the application. The usabilty test showed, that our application does not have any major usability problem and that the context management characteristic of our system improves the users' acceptance.

2. State of the art

2.1. Mobile services

In the last few years mobile technologies are booming. People want to be mobile and to do their business from everywhere, with as less effort as possible. Garter Inc. showed that worldwide 1.211 billion mobile phones have been sold in 2009. Smartphones sales totalled 172 billion units in the same year, a 23,8% increase from 2008 [7]. Such an increase in smartphone sales in the last years is especially caused by the availability of mobile Internet connections. The first version of GSM (1992) offered data rates of 9,6 kbit/s. Applications such as audio/video streaming were not thinkable, because of slow speeds and monochrome displays. It was neither possible to open HTML pages, so WAP, a mobile service for accessing special Internet pages (WML), was born [8]. Now UMTS (3G) offers rates of up to 7,2 Mbit/s (HSDPA) and WAP is harly ever used, because of the possibility to access and display Web 2.0 contents. In the near future (2012-2015) a new cellular network (4G) will be build up, which will enable speeds between 100Mbit/s and 1Gbit/s [9]. This advanced communication technologies opened new possibilities in providing mobile services. A new market of mobile devices raised up which influenced the hard- and software branch. Beside smartphones, notebooks and netbooks, mobile communication nowadays can be found also in many other artifacts of everydays life. Premium cars, for example are equipped with wireless communication systems for news, weather, road conditions, navigation and Internet connection for browsing [9].

Fast Internet access and modern hardware allow to port desktop applications on mobile systems. Nevertheless mobile applications have to be designed in another way than their desktop twins. Making devices portable generally means, making them as small as possible, but with the most possible integrated features. This leads to the availability of fewer resources (memory and battery), a smaller screen and limited input devices. Developing mobile applications, one has to deal with these issues: especially the user interface has to be designed in a way that it is clear and does not have too many options on a single screen. Another important issues is not to have too many background tasks, that consume memory and battery. In a nutshell a mobile application will not have the same functionality, as a desktop version. Studies of the Standish Group International, Inc have shown, that in software only 20% of the available features are often or always used by users. 64% are rarely or never used. A challenge is to integrate such functionality which the user really needs.

Since nowadays many mobile applications require online access, the availability of an Internet connection is an important aspect. Many mobile devices can establish Internet connections through wireless LAN or through the telephone network. Unfortunately open WLANs are rarely available and provider contracts which permit data connections

are not so usual, especially in South-Tyrol and for tourists which have the SIM card of their domestic provider.

2.2. Context-Aware Systems

"Context is any information that can be used to characterize the situation of an entity. An entity is a person, place, or object that is considered relevant to the interaction between a user and an application, including the user and applications themselves." (Dey, 2001) [17]
In his book *Mobile Design and Development* Brian Fling makes a distinction between two interpretations of context [5]:

- *Context* (with capital C) is how people understand, feel, interprete, live their circumstance. For example: *Being in a 1000 year old church* or *reading a book about that church* will enhance peoples experience. In both situations the person will derive value from the activity, i.e, Context, but there is a huge difference in how the two Contexts are influencing the understanding and the feelings of the user.

- *context* (with lowercase c) is the mode, medium, or environment in which a task is performed or the circumstances of understanding. Consider the previous example with the church and "location" as context dimension. For *"being in a 1000 year old church"* the value for this dimension is "in the church". For *reading a book about that church* the context-value will be the location where the person reads the book, i.e., in a train, in a park, at home, etc.

For our context-aware system, the second interpretation of context (with lowercase c) is more suitable, i.e., we considered context dimensions such as "companion", "knowledge of surroundings", "weather", "budget", etc. All contextual conditions we used can be found in section 4.1.1. Context (with lowercase c) can further be divided into three categories [5]:

- *Physical context* is the present location of a user and has an influence on her actions. A task might be executed differently, regarding whether she stays at home, in the bus, in the office,... There exist places where one feels and acts public, where many people are watching, i.e., the bus, the train and others where people feel private, like at home, with friends, etc.

- *Media context* is the present device of access. Different media, i.e., newspaper, TV, smartphone, provide different levels of value in specific situations. Newspapers provide plenty of information, but this information is mostly of yesterday. News services on smartphones do not have such a denseness of news, but are therefore up to date.

- *Modal context* is the present state of mind, which influences people in performing their tasks. Driven by needs and desires people make choices for accomplishing

goals. On a single day a person has to make thousands of minor decisions, which mostly are made automatically, but they have to be made: "Should I go left or right? Should I cross the street or not?". Each of those decisions is influenced by the current state of mind and has also an influence back to it.

It was shown that context is something very dynamic. Values for context dimensions can change over time. Consider for example "companion": during a single day, while meeting different people it changes value several times: "boyfriend/girlfriend", "alone", "family", "colleagues", etc. The same is for "weather", "temperature", "season", etc.

2.3. Recommender systems

"Personalization is the ability to provide content and services tailored to individuals based on knowledge about their preferences and behavior." [16] Recommendations are personalised suggestions, also called items, generated by a recommender systems using information about the user preferences and the preferences of other users (community).

The core computational task of a recommender system is to predict the subjective evaluation a user will give to an item [10]. According to Robin Burke, recommender systems have been divided into the following four main categories [14]:

1. **collaborative-filtering**
   As stated above, users can give feedback about how much they liked a recommended item. Collaborative Filtering (CF) systems use these user-item-ratings in order to predict how likely a user with similar preferences would be satisfied with the same item [11].

2. **content-based**
   Also content-based systems are based on the users' feedbacks, but when requested to provide a recommendation to a target user they consider only the preferences of that user and not those of others. The system considers the items that the user liked in the past and compares their description with items that might be suggested. If it founds similarities, then the item is nominated for being recommended [12].

3. **knowledge-based**
   A tipical issue of CF and CB recommender systems, is the so called "cold start" [18] or "ramp-up" [15] problem, where systems cannot make good predictions until there is a large number of users whose habits are known. Knowledge-based recommender systems avoid this problem. They do not depend on user ratings, but on a knowledge structure to make inferences about the users' needs and preferences. An important knowledge-based technique that is exploited within recommender systems is case-based reasoning (CBR) [13]. All generated recommendation sessions are stored in a case-base. New recommendations are generated, by retrieving similar recommendation sessions out from that base [10].

4. **hybrid**
   Hybrid systems combine two or more techniques in order to gain better performance with fewer limitations of each approach [10] [14].

2.3.1. Context-Aware Recommendations

Our research project (ReRex) implements a context-aware collaborative-based recommender system. It is an extension of a classical collaborative filtering system. Classical CF systems estimates the rating function R for (User, Item) pairs that have not been rated yet by the users [20].

$$R : User \times Item \rightarrow Rating$$

It returns the k highest-rated items for a user. These k items are those that the system considers as most valuable for the user. Such systems are called traditional or two-dimensional [20]. As stated in section 2.3, CF is based on user ratings for items that help to predict the ratings of like-minded users. In many domains, i.e., tourism, this definition is not sufficient to obtain appropriate recommendations. Users interests might be relatively stable, but the decision, on whether they are going to visit a point of interest or not, depends on many additional varying factors, i.e., on context [19] [21]. Consider for example the recommendation of a skiing event in summer. It is a very poor recommendation since most people go skiing in winter. In contrast to CF recommender systems, a context-aware recommender system is able to detect such a situation and would not provide an event like that. CF doesn't consider contextual conditions in its recommender algorithm.

As mentioned earlier ReRex implements context-aware collaborative filtering, therefore the rating function R, stated above has to be extended, by adding context:

$$R : User \times Item \times Context \rightarrow Rating$$

where User and Item are the domains of users and items respectively, Rating is the domain of ratings, and Context specifies the contextual information associated with the application [20]. For our used contextual conditions, please see section 4.1.1.

The choice of a context-aware recommender system was related to the specific tourism domain. As stated in section 1.3, the application was developed to be used by tourists that are new to Bolzano and never used the system before. Such real-world scenario rises the so-called "cold start"' problem [18], described in section 2.3, where systems need to make rating predictions without any information about the users tastes. Therefore, we were limited to use a non-personalized context-aware recommender system. The above rating function shows that context has an influence on the rating of an item (likelihood to visit a POI). Therefore, this rating of an item can be decomposed into several components (summands). One sub-ratings, $\overline{i}$, provides the default rating prediction and is computed as the average item i rating in the database. Each other sub-rating gives information how much a given contextual condition influences the final rating for an item. Therefore

the rating prediction for an item i is computed as:

$$Rating = \bar{i} + \sum_{j \in C} context_j + \sum_{j \in C} context_{ji}$$

where $context_j$ is the global influence of contextual condition with index j (the same for all items), and $context_{ji}$ is the influence of contextual condition j when predicting the rating for the item i. Only those contextual conditions are considered that have been enabled by the user. The above described linear model is learned using gradient descent method [22]. A detailed description of the method is out of scope of this thesis.

To train the context-aware rating prediction model we needed a context enriched rating data set. For determining an initial set of (user, item, context) ratings we created a web interface, shown in figure 2.1 where users can rate context influences on given items. The quality of the recommender system depends on the quality of data. Usually the more ratings are in the data set, the more accurate are the predictions. Because of a high number of contextual features we needed even more data, i.e., ideally each item should be rated in all contextual conditions by as many people as possible. We asked our colleagues to enter as many ratings as possible and at the end had around 1100 responses.

The context-aware system was compared with a popularity based algorithm. Popularity based recommender systems recommend the items that received on average the highest rates by all the users, in all the contexts.

2.3.2. Real-Time Recommendations

Real-Time Recommendations refer to a characteristic of *context* and *user preferences*: they may change over time. A real-time recommender system is able to react on such changes, by provide an updated recommendation list. Updates can be the following:

- **add**
  an item is added to the list of recommendations
- **delete**
  an item is deleted from the list of recommendations
- **replace**
  an item is replaced with another from the list of recommendations
- **orderchange**
  the order of items changed in the list of recommendations

Naturally it is not appropriate for a system to make such updates without asking the user for permission. She might not be willing to change something. It is useful to ask the user if she wants to accept or reject a change.

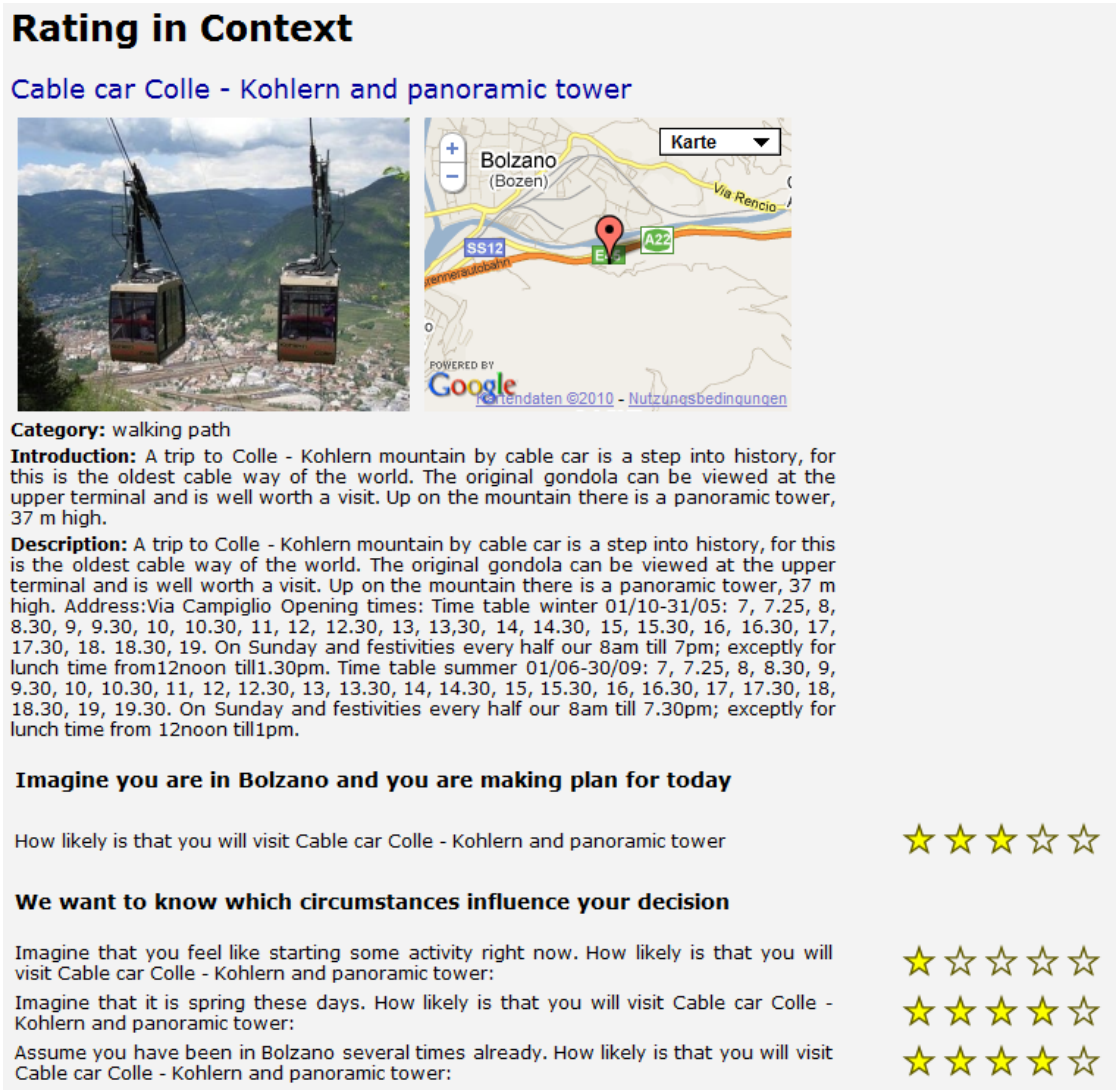

Figure 2.1.: Webinterface for retrieving user-item-context ratings

3. Problem description

We wanted to build a mobile application, such that is able to access a real-time context-aware recommender system (ReRex) in order to test whether the context actually can increase the usability of the system. Our hypothesis was, that context will definitly make the city guide more useful. In order to prove this we created two different systems, one where users can specify contextual conditions and another where they can't. By different usage scenarios, described in chapter 5, users should try out both systems in order to determine which one provides better recommendations.
Consider the following example: a guided museum tour is suggested which takes 4 hours.

- *How likely is that a couple would take this tour on a rainy day?*
  Probably they will do it, because of the bad weather.

- *What about a family with 2 children on a sunny day?*
  The children might get bored soon and on a day with good weather they would prefer to go swimming. Probably they won't do the tour.

There are two examples of contextual dimensions in this example: companion and weather. A recommendation is more or less appropriate, depending on the values of these dimensions. A RS that is not context-aware would suggest the tour in both situations, because the user likes museums. A context-aware recommender system would suggest the tour only in the first situation because it knows, that on good weather people and especially children wont be happy to take a museum tour.

4. Technical Approach

4.1. Human-Computer Interaction

4.1.1. User Profile and Context Model

As mentioned in section 2.3, a context-aware recommender system needs user- and context-features for making good, personalised suggestions. The application provides two views, shown in Figure 4.1, where these can be specified.

On the profile page, the user can specify characteristics and interests. Caracteristics are age and gender (Figure C.2, Appendix C). Interests are categories, such as *Museums*, *Castles*, *Events*, etc., to which users can give a rating from 1 to 5. Therewith they express how interested she is in each single category.

On the context page, the user can specify its contextual situation: he can enable and disable context, by on/off switches. A context which is disabled, isn't considered by the recommender system. After selecting a context, all its possible values are shown and one value can be selected (Figure C.1, Appendix C). For some contextual dimensions the user cannot specify the value, i.e., Distance to POI, Temperature, Weather, Weekday,... The recommender system obtains the value for them by accessing weather services, location APIs, etc. Such contextual conditions can only be switched on or off, depending on whether they should be considered by the recommender system or not.

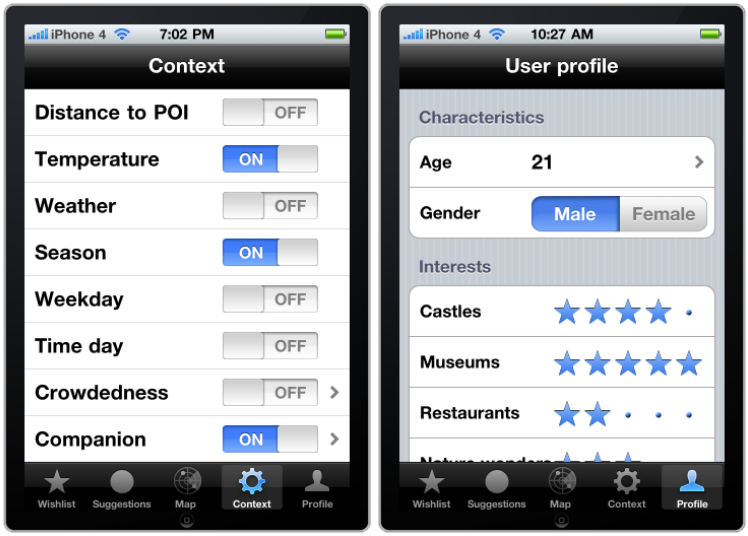

Figure 4.1.: Context and User Profile Settings

In our project we used the following contextual conditions:

Distance to POI
- Far away
- Near by

Temperature
- Hot
- Warm
- Cold

Weather
- Sunny
- Cloudy
- Clear sky
- Rainy
- Snowing

Season
- Spring
- Summer
- Autumn
- Winter

Companion
- Alone
- Friends/Colleagues
- Family
- Girlfriend/Boyfriend
- Children

Time day
- Morning
- Afternoon
- Night

Weekday
- Working day
- Weekend

Crowdedness
- Crowded
- Not crowded
- Empty

Familiarity
- New to city
- Returning visitor
- Citizen of the city

Mood
- Happy
- Sad
- Active
- Lazy

Budget
- Budget traveler
- Price for quality
- High spender

Travel length
- Half day
- One day
- More than a day

Means of transport
- Car
- Bicycle
- Pedestrian
- Public transport

Travel goal
- Visiting friends
- Business
- Religion
- Health care
- Social event
- Education
- Cultural
- Scenic/Landscape
- Hedonistic/Fun
- Activity/Sport

4.1.2. Points of Interest Suggestions

Having specified the above mentioned information (context and preferences) the system can be asked for generating a suggestion list, shown in Figure 4.2.

The list includes a few items, to not overload the user with excessive information. As shown in Figure 4.2 a list item is designed in a way that a user can have a first idea of the relevance of the recommendation by only looking at it, without a detailed view, by tapping on it. The most important information is the items name, its picture and category. The rating of 1 to 5 stars, computed by the recommender system, is the prediction of how likely a user might be interested in the recommendation item and should help her in making her decision. Some list items provide additional information, so called context updates or explanations. They are labelled by a clock icon, which shows if the item could be more or less interesting at that particular moment, because of a contextual condition. In Figure 4.2 *Messner Mountain Museum* is labelled with a green clock icon and an arrow which points up. It means, that this recommendation might be very interesting at the moment, because of one or more contextual conditions. In the detail view of that item, there will be found an explanation why this is the case

(Figure C.3, Appendix C).

If the user is not satisfied with the items in the recommendation list, then she can do one of the three following actions:

- Manually update the list by tapping on the refresh icon at the top left corner. This causes the iPhone to ask for a new recommendation list, but it's not sure that some new recommendations are computed.
- Play with context settings and interest ratings (Figure 4.1), so that the recommender system might provide other recommendations.
- Browsing all the available items by tapping on *more* on the top right corner. A list of categories will appear. The user can select one of them and obtains all available items of that category (Figure C.4, Appendix C).

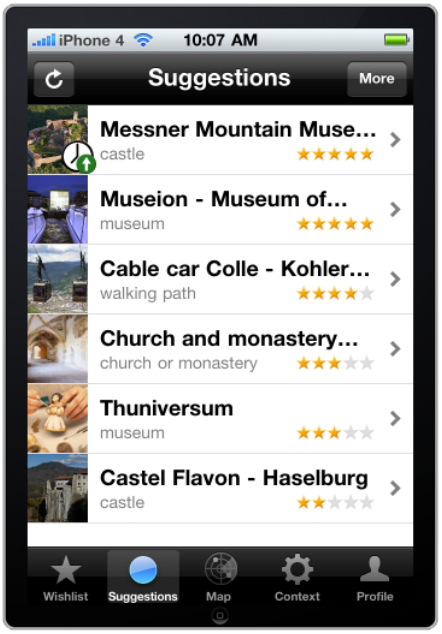

Figure 4.2.: List of recommendations

4.1.3. Points of interest

A point of interest is described by several properties - some have been explained in section 4.1.2. They are displayed in the details view of a POI which is composed by four pages (Figure 4.3). On the *Info* page, there are the title, a small picture, a short description and a rating provided by the recommender system. Additionally there is a button to add the point of interest to the wishlist - this will be explained in the next section. The *Details* page contains a larger picture and the whole description. On the

third view there is a map, provided by Google. A purple needle shows the POIs location and a blue point represents the users current position (only available if the iPhones location services are enabled).

As stated in section 2.3, recommender systems need users' feedbacks, saying whether the users liked the POI or not. Using these ratings, they are able to improve future predictions for users. On page four, called *Feedback*, the user can rate the item (from 1 to 5). Figure 4.3 shows for that particular POI only one rating component, called *General*, but for other items there might be more of them, i.e., "How much do you consider this POI on a rainy day", "How easy was it to reach this POI with public means of transport", etc.

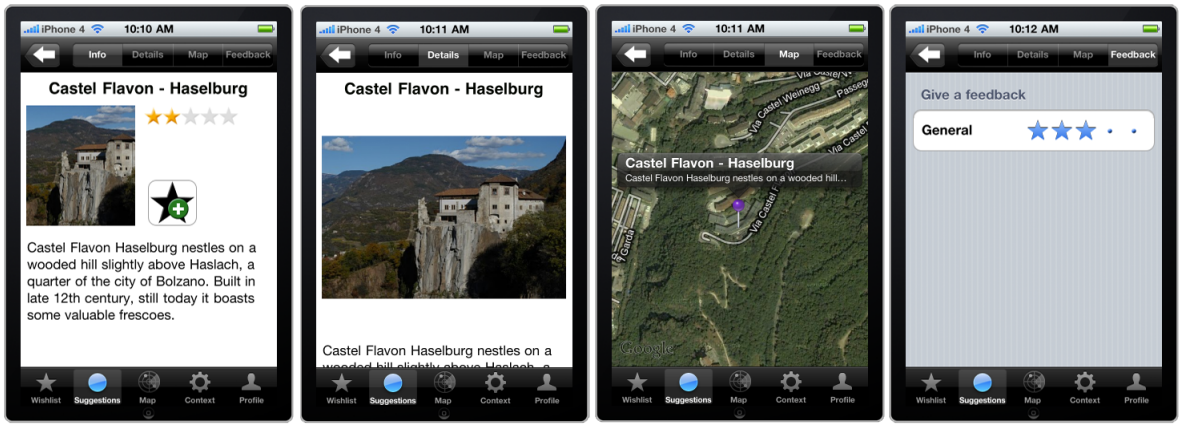

Figure 4.3.: The four detail views of a POI

4.1.4. Wishlist

If a user likes a recommended POI and wants to put this in her itinerary, she can add it to the wishlist. The wishlist can be described as a list of favourite POIs. It looks very similar to the suggestion list, but this will not change automatically. The user maintains it: she can add and delete items and change their ordering. Figure 4.4 shows that two items have a clock item nearby. The green icon with the arrow upwards was already explained in section 4.1.2. *Castel Flavon - Haselburg* has a red clock icon nearby with an arrow that points downwards. It means that there are one or more contextual conditions, which influence the recommendation of this item negatively. An item with such an icon is not recommended by the system, but since the user has it in her wishlist it won't be deleted automatically.

In the details view of a POI, in the *Info* page, there is a button *Add to wishlist* displayed as a star symbol with a plus in it. By pressing this button the user will be asked for a confirmation and then the item is added to the wishlist. A similar behavior is implemented for the deletion of an item from the wishlist. In this case the user has to press on the same star symbol, but this time there is a minus in it. After confirming the deletion, the POI will be removed from the wishlist (Figure C.5, Appendix C).

The order of wishlist items can be changed by pressing on *Edit* at the top right corner (Figure 4.4 and Figure C.6, Appendix C).

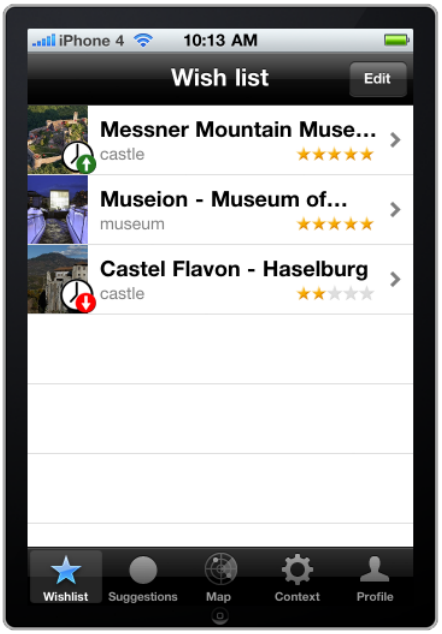

Figure 4.4.: Wishlist

4.1.5. Real-Time Changes

Till now we do not have introduced the real-time feature, which the application is also able to deal with (described in section 2.3.2). The recommender system might propose changes in the wishlist: add or delete items. Such proposals can be triggered by severeal factors. It can be by a modification of the user preferences or more likely because context changed. As figure 4.5 shows, the user has the possibility to review all changes the system proposes and accept or reject them individually. Pressing on *done* her decisions will be applied on the wishlist. If she do not want to receive any further real-time notifications, she can press on *Never ask me again*.

Changes in the wishlist are not the only ones the recommender system may provide: a second type of updates are context predictions. The recommender system has the possibility to forecast context settings, as for example: *ReRex predicts that you are travelling by bus* or *ReRex predicts that you are a budget traveler, since you travel with your family*. The user can review this predictions and decide whether to accept or decline them. The notification message is shown in Figure C.7 in Appendix C.

Notifications for real-time changes do also arrive if the application is not currently browsed by the user, but it is only running in the background. Figure C.8 in Appendix C shows how such a notification will look like. By pressing *Close* it will be ignored, but

pressing on *View* the application will be brought to foreground and it will allow to accept or reject the change.

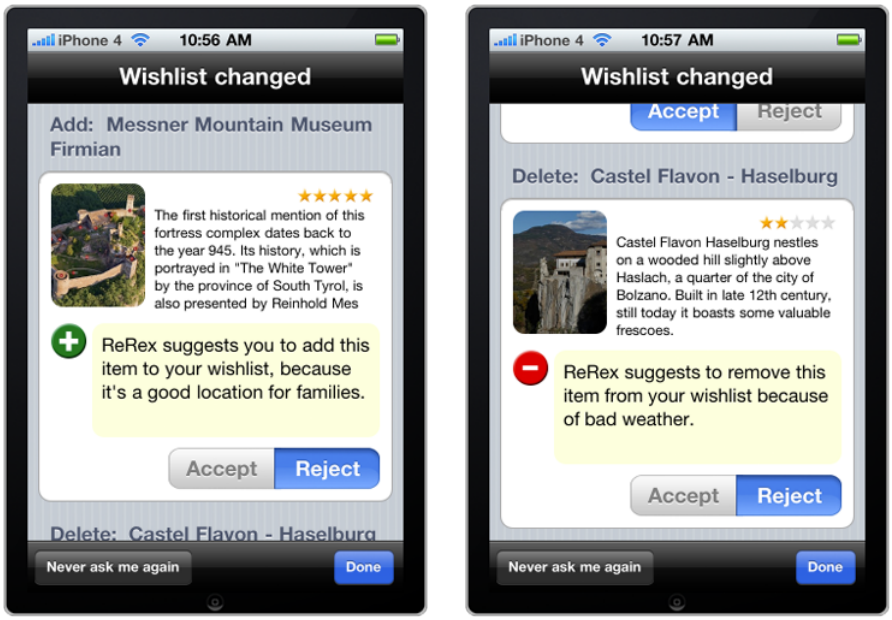

Figure 4.5.: Real-Time notification alerts for wishlist changes (add and delete proposals)

4.1.6. Map management

On the map are shown all wishlist and suggestionlist items. Wishlist items are marked with a red star, suggestionlist items with a green ball. Additionally the user can see her position on the map as a blue ball. This is only possible if the iPhones location services are enabled. The map is helpful for determining distances between items and the user's actual position and for planning an itinerary through the points of interest.

Figure 4.6.: Map view, showing POIs and user location

4.1.7. Application settings

For making the application more flexible we integrated three additional options. These cannot be found directly in the application, but in the *Settings application* of the iPhone. There is a subsection called *ReRex* (Figure C.9, Appendix C).

1. **Server URL**
   There one can specify the URL of the webservice. So one can dynamically switch between recommendation services which implement the same communication protocol. In an official release of the application this option will not be there, otherwise users might be confused.

2. **Reset Application**
   This option resets the application to its initial state. User profile, context, recommendation list, wishlist, etc. will be cleared or set to default.

3. **Real-time changes**
   By disabling this option the application will no longer notify users about real-time changes. If it is enabled, the user will get notifications on updates, provided by the recommender system (Figure 4.5 or Figure C.8, Appendix C).

4.2. System architecture

We divided the implementation of the whole project in two parts: server and client. The server contains all the recommender logic and the needed databases. The client implements the GUI on the iPhone and it is responsible for displaying recommendation data in a proper way and to allow users to make their input.

All points of interests, context features, categories, etc. are stored in a PostgreSQL database running on the ReRex server (rerex.inf.unibz.it). Additionally an Apache Tomcat Webserver (Javabeans, JSP, Servlets) provides all the recommender logic as a web-service for the iPhone client. Figure 4.7 shows the major system components:

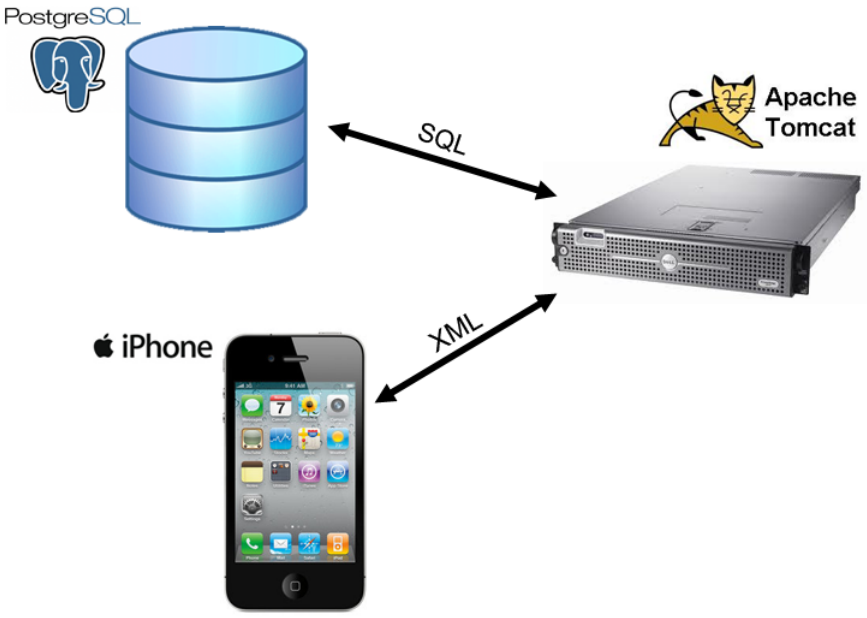

Figure 4.7.: PostgreSQL and Apache Tomcat provide server side recommendation services, the iPhone client is able to access them.

Initially we thought to give all recommender logic to the iPhone, so we would not need a server - a single point of failure - and no Internet connection. But this has some big disadvantages. As stated in section 2.3, recommendation strategies are based on very complex algorithms and they need a lot of computational time and memory. Additionally some of them need the feedback of other users. This is not feasable on a mobile device. The next problem of such a solution would be a redundant and not maintainable database (POIs, categories, contexts, etc.) on every phone.

In order to provide the whole functionality, described in section 4.1 different frameworks had to be used. In Cocoa Touch such frameworks are collections of classes, like packages in Java.

- **Foundation.framework**
  contains all standard classes of Objective C, such as NSArray, NSString, NSObject, NSXMLParser, etc.

- **UIKit.framework**
  contains all standard user interface elements, such as UIViewController, UITabBarController (Navigationcontroller), UIImage, etc.

- **CoreGraphics.framework**
  contains colors, fonts, shapes and other elements for customizing the user interface: CGColor, CGFont, CGGeometry, etc.

- **CFNetwork.framework**
  contains classes for network access, such as CFNetwork, CFNetworkError, etc.

- **CoreLocation.framework**
  contains classes for determining the devices current position and accessing GPS services, such as CLLocation, CLLocationManager, etc.

- **MapKit.framework**
  contains the Google Maps API, for displaying locations on a map (Annotations): MKUserLocation, MKAnnotation, MKPinAnnotationView, etc.

4.2.1. Database structure

In this section we will show the most important relations of our database.

- **POIs**
  All available points of interest are stored in this table. Each POI is composed by id (PK), GPS coordinates (eastcoord, northcoord), title, description, image URL, location and category. The category is a foreign key to the categories table.

- **Categories**
  This table contains all categories. Its major attributes are id (PK) and the name.

- **Context Factors**
  Context factors are all available contexual conditions, i.e, (dimension, value) pairs. Primary key of this relation is an ID.

- **Contextual Rating**
  This relation stores ratings for (user, item, context) tuples. It realizes the rating function described in section 2.3.1 and is therefore the base for the context-aware collaborative-filtering recommender system.

- **Userlog**
  This table is for logging each single request, from any client. Its attributes are id (PK), userID, the clients XML and the time of the request.

4.2.2. Webservice

A webservice is a kind of application, which is typically delivered over HTTP (Hyper Text Transfer Protocol) [23]. Therefore it is a distributed software system based on a client-server architecture. The biggest advantage of this technology is its platform independence and language transparency. The client of a webservice can use its own application model and OS. This can be totally different from that of the server (ex. server: Debian Linux, Java - client: iPhone OS, Objective C). They only must have a common interface (protocol), for data transmission. The protocol is typically XML based, as in our application (see section 4.2.3). Normally webservices have an additional layer between HTTP and the application objects, basically SOAP (Simple Object Access Protocol), REST (Representational State Transfer) or similar protocols [23]. They serialize application data to XML for transmission over HTTP and deserialize it on the receiver. On both, sender and receiver, special libraries have to be available which support such protocols. Unfortunately the iPhone OS currently does not provide any of these libraries, therefore we could not use an intermediate SOAP or REST layer. We built our own methods on client and server to serialize objects to XML and to deserialize, i.e., parse, XML documents.

Web technologies, based on HTTP, have for mobile devices another advantage: since HTTP is a stateless protocol, they are stateless in their core (without session management, etc.). Mobile devices might often change their position and signal strenght will vary, therefore Internet connection often might get lost. Stateful protocols like CORBA, RMI, etc. will definitely have problems if signal strength is low, because they have to maintain the connection, while in HTTP the connection is closed after request and response are made. In addition we need to transmit small data packets, therefore an HTTP web component, based on servlets, was the best solution for us. It offers an interface, which parses the XML sent by the iPhone via a POST method request. For analyzing the users' behaviors all the requests are logged in a database. The server analyses the client's XML and creates a new XML document, that is sent in the response. This contains all the information that the client has to know: recommended items, wishlist items, contexts, contextual changes, proposes in wishlist changes, categories and only if requested all items of a given category.

4.2.3. Communication protocol

XML is used as communication protocol and language. XML is very flexible and we defined our own document type, presented in Appendix B.1 and B.2. As explained in section 4.2.2, iPhone OS does not provide other languages like SOAP, JSON, REST, etc., so the only solution was to create our own XML parser implementing a custom XML-based communication protocol. We used the NSXMLParser class from iOS 4 Cocoa framework [24].

A very important feature we introduced in the XML documents is version management. This is important because we plan to release our application in the Apple Store and therefore make it public for everybodys usage. Sooner or later the XML document

and the application will be extended and newly released, so it could not be anymore compatible with the old version. The versioning will allow to notify users about a new release of the application and advice them to update (Figure C.10, Appendix C).

```
<rerex version="1.0">...</rerex>
```

Communication happens in two ways, from client to server and vice versa. Therefore we needed two protocols for data exchange. They are structured in a similar way, but with significant differences. In the following two subsections we wanted to show the specifications of our protocols and how to use them.

4.2.4. Server to Client Communication

Suggestions

As explained in section 2.3, the major task of the recommender system is to provide recommendations, i.e., a list of items or suggestions. For each item, beside general characteristics about it, like title, description, picture and GPS coordinates, the server provides additional information: a unique identifier for an item (id) and the ids of the categories the item belongs to, i.e., 2=Museums, 3=Castles. Additional it provides the average rating of the item, provided by all users and optional contextual explanations (contexttext), i.e., why such item has been recommended to a user. The *userfeedback* element specifies the type of rating provided; in this example it is "General", meaning that it is the overall average of the users' ratings.

```
<suggestions>
    <item id="1">
        <categoryID>2</categoryID>
        <categoryID>3</categoryID>
        <title>School Museum</title>
        <description>The School Museum...</description>
        <shortdescription>...</shortdescription>
        <pictureURL>http://www.peerweb.it/sm.jpg</pictureURL>
        <longitude>11.34907</longitude>
        <latitude>46.4982</latitude>
        <rating>3</rating>
        <userfeedback>General</userfeedback>
        <contexttext type="high">This could be interesting for
            you, because it is rainy weather.</contexttext>
    </item>
    <item id="2">...</item>
    ...
</suggestions>
```

Wishlist

The wishlist is composed similarly to the suggestion list. The major difference there is, that it is managed by the user and the real-time feature comes into play: the recommender system might propose to the user to change her wishlist, i.e., to add or delete items. This information is given in the following XML.

```
<wishlist>
    <item id="1">
    ...
        <change type="delete">
            <reason>This POI is not anymore recommended since
               you are travelling with children.</reason>
        </change>
    </item>
    <item id="2">...</item>
    <item id="3">
        ...
        <change type="add">
            <reason>This POI is highly recommended to you,
               since the weather is good.</reason>
        </change>
    </item>
</wishlist>
```

Context predictions

Another type of real-time updates are context predictions. They might be used by the server to propose the change of the value for a contextual condition, i.e., ReRex predicts that you are travelling by bus.

```
<contextchange name="Means of transport" value="Public
    transports">
    <message>ReRex predicts that you are travelling by bus.</
       message>
</contextchange>
```

Contextual dimensions

We created the system in a dynamic way, such that the server can specify which contextual conditions should be available on the iPhone. In order to inform the client about what contextual conditions can be used, the following XML has to be provided by the server. The advantage of this approach is, that there is no need to newly release the application, if one wants to introduce new context dimensions.

```
<contextlist>
    <context name="Weather">
        <contextvalue dbid="12">Rainy</contextvalue>
        <contextvalue dbid="12">Sunny</contextvalue>
        ...
    </context>
    <context name="Companion">...</context>
    ...
</contextlist>
```

Categories

The server provides all categories to the client. A category has an id and a name. For user friendlyness purposes there can also be specified the plural name of a category, i.e., Castles.

```
<categories>
    <category id="1">
        <name>Museum</name>
        <nameplural>Museums</nameplural>
    </category>
    <category id="2">...</category>
    ...
</categories>
```

The last extract from the server to client communication protocol is for submitting all items of a given category.

```
<itemsForCategory id="2">
    <item id="10">...</item>
    ...
</itemsForCategory>
```

4.2.5. Client to Server Communication

User profile

This section of the XML protocol contains all the available information about the user: its ID, its current position, its characteristics and interests.

```
<user>
    <deviceid>J1231S-7391AJKS-82193</deviceid>
    <age>22</age>
    <gender>male</gender>
    <interests>
        <castles>3</castles>
```

```
        <museums>3</museums>
        <restaurants>4</restaurants>
        <naturewonders>5</naturewonders>
        <events>5</events>
    </interests>
    <location>
        <longitude>12.312311</longitude>
        <latitude>13.122211</latitude>
    </location>
</user>
```

Contextual conditions

Beside the user profile, the server has also to be informed about which contextual conditions are enabled and what are their values. "Auto" means, that it is a contextual condition whose value is computed by the server. Dbid is the id of the contextual condition in the database.

```
<contextlist>
    <context name="Companion" on="false" value="Family" dbid="
        12" />
    <context name="Temperature" on="true" value="auto" dbid="
        auto" />
    ...
</contextlist>
```

Suggestions

The suggestion list is composed by all the items that currently are recommended by the RS. They are always sent to server in order to inform him of eventual user ratings (feedback).

```
<suggestions>
    <item id="1">
        <userfeedback name="General">3</userfeedback>
    </item>
    <item id="7">
        <userfeedback name="General">0</userfeedback>
    </item>
    ...
</suggestions>
```

Wishlist

The wishlist is completely managed by the user. It has to be communicated to the server, so that real-time updates can be provided.

```
<wishlist>
    <item id="3">
        <userfeedback name="General">2</userfeedback>
    </item>
    <item id="4">
        <userfeedback name="General">5</userfeedback>
    </item>
    ...
</wishlist>
```

Categories

In order to cause the server to send all items of a given category, the following tag can be specified.

```
<requestItemsForCategory id="2" />
```

4.3. Logical architecture

For creating an iPhone application, Apple provides a framework: the Cocoa Touch iPhone SDK. It contains the whole functionality needed for the ReRex application. Cocoa Touch strictly requires the use of the MVC (model - view - controller) pattern [25]. The model are objects which contain the data, i.e., the data structure. The view is responsible for displaying the needed information. The controller handles the user's inputs and performs actions on model and view.

In Cocoa Touch the view is implemented by the interface builder. This let you to create a view by just drag'n'drop UI elements into each other. The file is saved as *.xib. Each view has an associated controller, a subclass of the UIViewController, in which all actions are handled. The connection between user interface elements of the view and the controller is made by so called Outlets. Outlets are just pointers to view elements, that have been created with the interface builder. The model is implemented by simple classes, mostly derived from NSObject, the root of the class hierarchy.

There are many different models, views and controllers but nevertheless one needs a point where all converges, this is the *Application Delegate* class. It is instanciated when the application starts and deallocated when it terminates. It stays in memory if the application moves to background. In addition, this class can be accessed from every other class, without the need of passing instance variables of it. This makes it obvious to let this class be the root of the model-objects. So they can be easily accessed from every other class, in any situation. Since the iPhone has only a small screen, it cannot display all the implemented functions in a single view. Many views and their

corresponding controllers are needed and the application switches between them. It was useful to define the *Application Delegate* to be the root of all controllers, so that the view switching can happen at any possible view, in any possible controller.

In the implementation of the application there have been used two kinds of controllers: *View Controllers* and *Navigation Controllers*. View Controllers are responsible for controlling a view, i.e., to perform and handle actions on it. Navigation Controllers deal with the switching between two or more views. The main navigation controller in the application is the Tab Bar Controller, which contains the main menu (Suggestions, Wishlist, Map, Context, Profile). Also for list menus (tables), there are needed navigation controllers. Figure 4.8 shows how the view and navigation controllers are organised to provide the applications functionality. In this figure are not shown all the controllers, but only the most important ones for understanding the structure of the application. Furthermore a well organised data model is needed, which can be accessed by this controllers. The main components of the used data model, are illustrated in Figure A.1, Appendix C.

As mentioned earlier, the iPhone application downloads data provided by the recommender system, but we decided to store this information also locally on the phone in a persistent way. This allows to use the application also offline, surely without getting any updates on recommendations. Model objects are kept in volatile memory and are deallocated after termination. To store them in a persistent way, all the objects have to be created in a special manner, so that they are serializable. When the application starts, they are retrieved from local storage and volatile objects are created (model). Before the application will terminate, the model is stored persistently back.

Another aspect of the data model is the maintainance of the various lists of POIs: Suggestion list, Wishlist and the list of all items per category. There might be a situation where items occurrs in more than one list. In that case the items would be handled as different objects, i.e., the user could give different ratings to them or add them both to wishlist. To solve this problem, we introduced a fourth list, which stays in background and contains all items the application knows. The other lists contain only references to items of that main list. So each POI is allocated only once.

A very important component is also the XML parser. It establishes an HTTP connection to the webservice in a thread, so that the user interface will not be blocked. It sends the client XML, defined in the DTD in Appendix B.2 via a POST request to the webservice. Then it retrieves the XML generated by the server, which is structured according to the DTD, shown in B.1, and parses it. After parsing all the updates, for instance new recommendation items, real-time updates, changed explanations for items, etc., are merged into the iPhones data model.

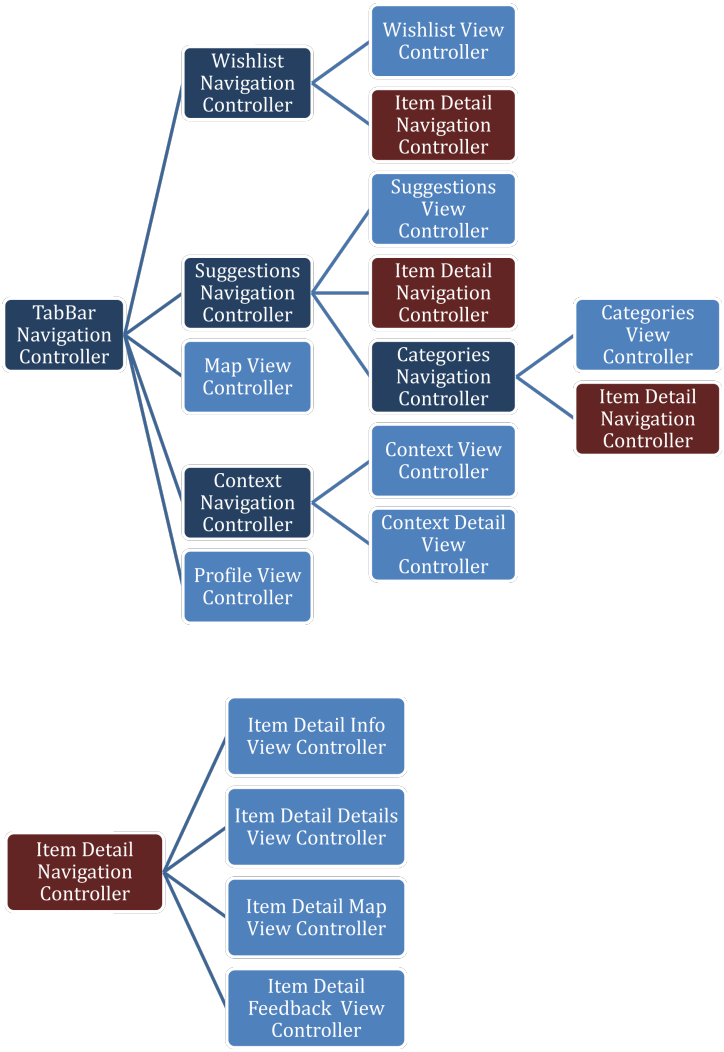

Figure 4.8.: Structure of the View and Navigation Controllers

4.4. Technical issues

For developing the context based recommender system different technologies have been used. On the client side the iPhone SDK was used. The implementation of the application started under iPhone OS 3.2, but in June 2010 iOS 4 was released officially by Apple, so the code was updated to this new OS. Some new features in iOS 4 were very helpful in developing the application and made it much more flexible.

4.4.1. Real-Time Notifications

The most important innovation was multitasking. But it is not real multitasking as known on desktop computers, which would consume the battery too fast. Applications can rather be put in background and then resumed from it. All applications which have been running once and closed by usage of the home button are in background. This is a freezing of the application in its current state. If it is opened again it switches from background to foreground, recovering the freezed state and starts running at the same point where it was closed before. A very important feature of iOS 4 is to let a backgrounded application execute some code. So every 10 minutes a request to the server side recommender system can be made for looking if there are updates or changes. If there is a change, a local notification will be triggered: it's an alert which informs the user about the change. The advantage of this solution is, that it has nearly no effect on battery performance.

The same could be reached also with iPhones Push Notification Framework. Surely it has the advantage that immediately when the recommender system on the server detects a change, it is pushed (send) to the iPhone. But it has two disadvantages and because of them we decided not to use this framework:

1. Push notifications are realised by a connection to Apples Push Notification Service (APNS), which is always opened and waiting for a push message. This is reducing the battery charge significantly. Push notifications therefore often are disabled by users, which lead to the effect that they will not be notified of real-time changes.

2. The server has to be implemented in a special way: it cannot send push notifications directly to the iPhone, but has to send them to Apples Push Notification Service which forwards them to the iPhone.

Additionally, the application sends every significant change in the user interface and its whole state to the server and asks therewith for an XML, which contains eventual updates. This approach is effective, because there are only small data packets which will be sent and received, i.e., it doesn't matter if we send only the changed settings or the whole state. What matters is, that it's easier for the server implementation as it is like that.

5. Usability Analysis

5.1. Research hypothesis

The main research hypothesis that motivated our project is that the context can increase the effectiveness of a mobile recommender system. In order to prove that we created two very similar versions of the same system, version A and B. System A is basically the same as described in chapter 4.1, but with two significant differences:

- We removed the context-awareness feature of the system.
  In practive, we removed the *context* tab in the main navigation controller. So, the user was not allowed anymore to specify contextual conditions for getting more appropriate recommendations, i.e., all context dimensions are disabled. Therefore it was like a not context-aware CF recommender system.
- We note, that if there is no mention of the context, we could not provide any explanation about why the context is influencing the recommendations. Therefore we also removed all the explanations for items' recommendations (Figure C.3, Appendix C).

In order to determine which system is more effective we stated the following hypothesis:

H: Version A (context-aware) is preferred to Version B (not context-aware) We expected that users prefer the context-aware version A of the system, compared to B which is not context-aware. The context-aware system provides recommendations that might be more appropriate to the users current circumstances. Nevertheless the not context-aware version might be easier to use, since the user must not specify contextual conditions and therefore must not understand that concept.

5.2. Evaluation Strategy

In order to test the above mentioned hypothesis we conducted a usability test. Test participants tried out both implementations of the system and executed two different usage scenarios (Figure D.1, Appendix D), with each variant of the system.

Usage scenario 1: Imagine you are living in Bolzano. Suppose that it is a cold, rainy Wednesday and you are alone. Select a single point of interest of your choice and add it to your wishlist.

Suppose now your parents are coming to visit you. If needed, revise your previous selection and add another point of interest to your wishlist.

Usage scenario 2: Imagine to be a tourist visiting Bolzano. Suppose that it is a sunny, warm afternoon and you are traveling with your girlfriend/boyfriend. Select a single point of interest of your choice and add it to your wishlist.

Suppose now that starts raining. If needed, revise your previous selection and add another point of interest to your wishlist.

The subjects were divided in four groups, each one receiving a different scenario and system combination in all the possible ordering. This was meant to give, on average, equal conditions to the two systems and counterbalance any learning effect.

G 1: Scenario 1 using system A and then Scenario 2 on system B.

G 2: Scenario 2 using system A and then Scenario 1 on system B.

G 3: Scenario 1 using system B and then Scenario 2 on system A.

G 4: Scenario 2 using system B and then Scenario 1 on system A.

The experiments were conducted during two days and were performed by 20 participants. The subjects were aged from 20 to 40 and 12 were either researchers or students in computer science. The experiments were performed in a room with not many people in it. We gave to all the subjects an iPhone 3GS with the running application and let them get familiar with it. As described above, for each usage scenario they should add two points of interest to their wishlist. From where they select these items was their choice: they could either use the recommended POIs or find them in the list of all items. In the context-aware version of the system (version A) they should also specify contextual conditions according to the usage scenarios, in order to get more appropriate recommendations.

During these trials of the system we logged all the major actions the users made. In the future we plan to analyse from where they took the points of interest. So, for instance, one can check whether they were satisfied with the recommendations the system provided or if they took many POIs from the list of all items.

In order to get a feedback on the usability of our system we asked the subjects to fill out a questionnaire. We used the *Computer System Usability Questionnaire* from IBM, which evaluates systems at a global level and at a detailed scenario level [4]. The scale of the questionnaire normally goes from 1 to 7. Such a wide range might make it hard for the subjects to state their evaluation. Therefore we reduced the range to 5, i.e., it goes from -2 (strongly disagree) to 2 (strongly agree). We adapted also the questions to our specific application domain and divided them into several categories.

First we asked some few general questions about the user, i.e., how often and in which area she uses mobile Internet. Additionally we wanted to know what kind of mobile device she is using. With this information we could understand whether the user is an expert in mobile Internet usage.

The main part of the questionnaire is divided into two parts. One section for each of the systems where we asked questions about their usability and effectiveness.

First we wanted to have some general feedback about the user interface.

Q 1: **It was simple to use this system.**

Q 2: **The interface of this system is pleasant.**

We wanted to know how easily users can find their preferred points of interest using the system and get information about them.

Q 3: **The organization of information provided by the system is clear.**

Q 4: **It is easy to find the information I needed.**

Q 5: **The system is effective in helping me complete the scenario.**

An important aspect of usability is also, how long does it take for a user to get used to an application, i.e., the learning aspect.

Q 6: **It was easy to learn to use this system.**

We wanted also to know the general opinion of the users about the system.

Q 7: **Overall, I am satisfied with this system.**

Q 8: **I like using this system.**

Q 9: **This system has all the functions and capabilities I expect it to have.**

According to our research hypothesis we wanted to know which system is more effective. Therefore we introduced two questions about recommended items.

Q 10: **I am satisfied with the suggested points of interest.**

Q 11: **I can effectively find interesting suggestions using this system.**

We put two open answers in the questionnaire, in order to give the users the possibility to describe aspects of the system they didn't like or they really liked.

Q 12: **Most positive aspects (if any).**

Q 13: **Most negative aspects (if any).**

In the questionnaire for system A, i.e., the context-aware one, we stated also some questions regarding context and contextual explanations.

Q 14: **I understood the benefit of using the contextual conditions.**

Q 15: **It was easy to specify the desired contextual conditions.**

Q 16: **I am satisfied with the provided contextual explanations.**

Q 17: **I believe that the contextual explanations are useful.**

Q 18: **The contextual explanations provided by this system are clear.**

At the end of the interaction with the two systems we asked which system the user preferred (Q 19) and which one suggested more appropriate points of interest (Q 20). The whole questionnaire is shown in the Appendix D (Figure D.2 and D.3).

5.3. Experimental Results

5.3.1. Mobile Internet Usage

The first section of the questionnaire contained questions about the familiarity of the users with mobile Internet, i.e., which devices they are using and if they are using mobile Internet. Our study showed that the majority (80%) of the subjects never or seldom used Internet on their mobile device (Figure A.2, Appendix A). This is less than we expected since the majority of them are computer scientists. Only 3 of the 20 participators own an iPhone and are therefore used to the style and interface of applications like ours (Figure A.3, Appendix A).

5.3.2. System A vs. System B

Questions Q1 to Q11 are the same for both systems, therefore they can be used to determine which system achieved better results in the usability test. We computed the average for each question and the comparison is shown in Figure 5.1.

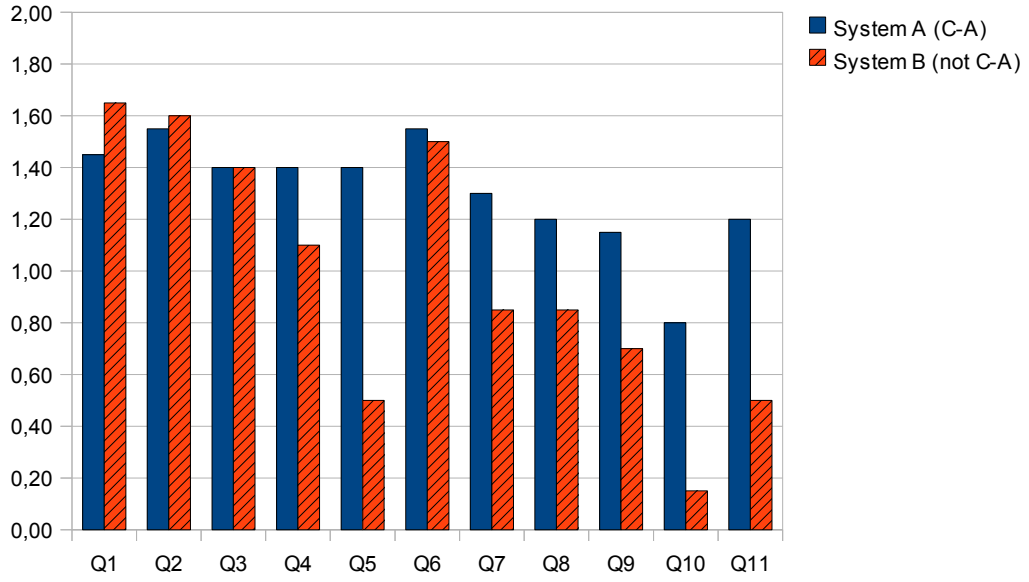

Figure 5.1.: Average answers to the usability statements Q1 to Q11 for each of the two versions of the system

Comparing the systems statement by statement, Figure 5.1 shows that most of the cases the context-aware version (system A) better scores than the non context-aware (system B). There are two exceptions: questions Q1 and Q2. This is reasonable, since contextual conditions make the system A more complex, i.e., the system B is easier to use. The average evaluation of the questions Q1 to Q11 showed that system A is slightly preferred (Figure 5.2).

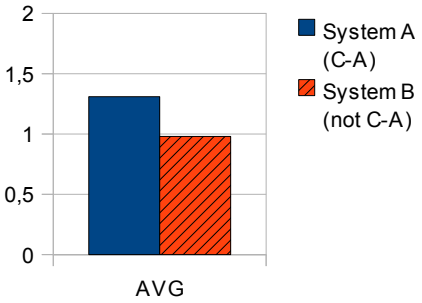

Figure 5.2.: Average evaluation, based on the answers to statements Q1 to Q11

In order to verify our main hypothesis, i.e., that users prefer the context-aware version A of the system, we conducted a statistic hypothesis testing for each one of the 11 statements. A two tailed test was conducted with the null-hypothesis $H_0 : \mu_1 = \mu_2$, where μ_1 is the average response to a statement in version A and μ_2 is the average response to the corresponding statement in version B. Alternative hypothesis is $H_1 : \mu_1 \neq \mu_2$. Since the sample size is small (20), we used a paired t-test statistics to verify our hypotheses. We considered a significance level of p = 0,05 as enough to reject the null-hypothesis. Table 5.1 shows the p-values for all statements. Confidence values, higher than 95% are highlighted in green.

Q1	Q2	Q3	Q4	Q5	Q6	Q7	Q8	Q9	Q10	Q11
0,163	0,772	1,000	0,186	0,005	0,772	0,035	0,130	0,025	0,019	0,035

Table 5.1.: Calculation of the p-values for statements Q1 to Q11

As explained above, both systems are very similar, therefore we accepted 6 hypotheses which are below the 95% confidence level. We could not accept the null-hypotheses for the statements Q5, Q7, Q9, Q10, Q11, thus these represent the differences in the two systems. Looking at the averages in Figure 5.1 one can easily see that for all these statements system A is evaluated higher.

Q 5: System A is considererd as more effective for completing the scenarios. Since the only difference between the two version are contextual conditions, we can conclude that the availability of context makes the system more effective.

Q 7: The user is generally more satisfied with System A than System B. There can be concluded that the availability of context satisfies the user.

Q 9: System A has all the functions and capabilities users expect it to have. System B provides obviously less functionality since it is a subset of A.

Q 10: Users are more satisfied with the recommendations of system A than with those of system B. Therefore the availability of context causes the recommender system to provide better recommendations.

Q 11: Using system A users can find more effective suggestions in which they are interested. Also this statement pushes the context-aware recommender system.

The last section of our questionnaire provides two questions (Q19, Q20) where the subjects should choose the preferred system. Figure 5.3 shows clearly that most of them preferred system A, i.e., the context-aware one. Additionally 95% of the participants stated that system A suggests more appropriate points of interest (Figure 5.4).

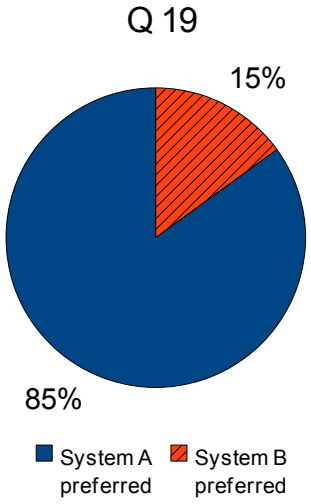

Figure 5.3.: Prefrences over the systems.

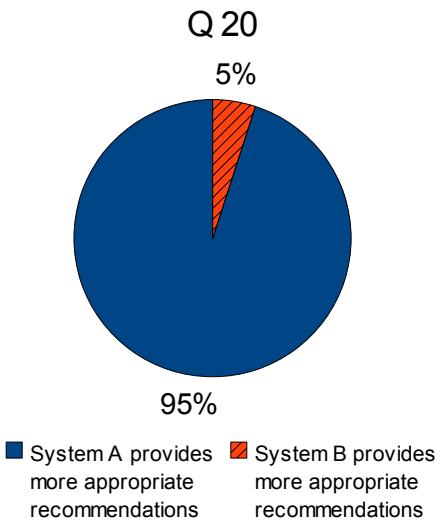

Figure 5.4.: System that suggests more appropriate points of interest.

These diagrams show that system A is the favourite. In order to proove that this result has not been obtained by chance, we conducted a χ^2-Test statistic, shown in Table E.3, Appendix E. The null hypothesis states that there is an equal distribution, i.e., half of the test participants prefer system A, the other half prefer system B. The χ^2-Test provided very small p-values: 0,00174 for Q19 and 0,00006 for Q20. These low values indicate that it is very unlikely that this sample is taken from a distribution where the two outcomes (prefer A or prefer B) have equal probability. Therefore we can state that system A is preferred by the users and also suggests more appropriate points of interests.

Finally we can accept our main hypothesis, stated in section 5.1:
Context can increase the effectiveness of a mobile recommender system.

5.3.3. Contextual conditions and explanations

In the questionnaire we included 5 questions, Q14 to Q18, about the understanding, appealing and usefulness of contextual conditions and contextual explanations provided in system A. The results of the users' responses to these questions is shown in Table E.2, Appendix E. On average these questions have a reply equal to 1,23 with a standard deviation of 0,97 on a -2/+2 scale. In conclusion, we can state that the users perceived the advantages of the context-aware system with respect to the non context-aware version.

6. Conclusions

6.1. Summary

In this project we wanted to test what is the benefit of using contextual information in a mobile recommender system. We created a mobile user interface on the iPhone which accesses a real-time context-aware collaborative filtering recommender system. In order to test whether contextual conditions can influence the recommendations in a positive way we conducted a user study. For this we created a second version of the system, where we removed all the context-management features, i.e., this second system offers recommendations suited to the preferences of the user but not adapted to the particular context of the user. The subjects tried out both systems and were asked to evaluate the systems using a *Computer System Usability Questionnaire*, which has been adapted by us. The analysis of the results showed, that the context-aware recommender system provides more appropriate recommendations and appeared more pleasant to the users. This outcome proves our hypothesis, that context can increase the effectiveness of the system.

The major problem we had to face in this project was the availability of points of interest data and user ratings. We got about 20 points of interest from suedtirol.info, but the problem was to get the ratings, i.e., user feedback, for these POIs in order to train the recommender system. Using a web-interface people could rate the items in specific contextual situations and at the end we obtained about 1100 responses. Using these contextual ratings the system was then able to provide points of interests to users - adapted to their current situation, circumstances, emotions, etc., i.e., context. Not context-aware systems ignore these conditions and may therefore provide poorer recommendations in certain situations. A further advantage of our system is the real-time feature. It is able to detect changes in contextual situations, informs the user about them and provide alternatives. Especially in the tourism domain this is very convenient, i.e., one would not like to make a mountain tour when bad weather is predicted. This can be exploited in a mobile device, which is always with the user, and can inform the user before it is too late.

In conclusion, our mobile application is a first step toward the implementation of a real service for the visitors of Bolzano; to inform them, in real-time, about what can be done at a specific time and place. Surely the system is not restricted to the use in Bolzano. It depends on available data, but in theory the system can be used also for other cities or even countries. Without any major change it can also be used for other application areas in which recommendations are useful, i.e., music recommendation.

6.2. Future work

Many users stated in the questionnaire that they really liked the user interface, but the system offers too few points of interested. At the moment, we tested the application only with a set of about 20 POIs. In reality the system might have hundreds or thousands of locations, events, museums, restaurants, etc. For the recommender system itself, huge amounts of data might not be a problem, but if a user wants to navigate through all the items of a given category, there will be too many in the list. To solve this problem the items could be displayed on different pages, showing only about 10 or 20 items. Another solution could be a search functionality, where users can specify keywords and corresponding items will be returned.

Furthermore until now we implemented just two kinds of wishlist-changes which may occur in real-time: items might be added and deleted. A future task would be to realize also the other two change types, described in Section 2.3.2: *order-change of the listitems* and *replace items by others*.

The context view of the existing application could be improved. Users like icons and pictures and the view is actually very technical. For each context feature and their values there could be introduced nice images, which help to understand what is meant.

As already mentioned in Section 4.3, the application is made in a dynamic way, such that the server can determine by XML what the iPhone should display. Recommendations, Context, properties of POIs, kind of feedback of POIs, categories, real-time messages and context changes are handled in this way. The only thing which currently is hardcoded is the user profile, composed by characteristics and interests. A future improvement could be to make also this dynamically settable by the server. As it is now, there is the disadvantage that every change needs a new release of the application in Apples App Store. This is also evident in the XML protocol definition in section B.2

Another interesting functionality would be an itinerary through POIs. Actually the system does not provide a feature, which suggests to the user the order of visiting items. It could be very useful for her to have a system, which leads her from one point to another. This itinerary could also be shown on a map, like a navigation system. The next step would then be, how to come to that point, i.e., what transportation means to use: the own car, the bicicle, public transports, etc.

References

[1] Toffler, Alvin. *Future Shock.* Bantam Books. 1970.

[2] Ricci, Francesco and Rokach, Lior and Shapira, Bracha. *Recommender Systems Handbook.* Springer. 2010.

[3] Swarbrooke, John and Horner, Susan. *Consumer Behaviour in Tourism, Second Edition.* Butterworth-Heinemann. 2006.

[4] Lewis, J. R. *IBM Computer Usability Satisfaction Questionnaires: Psychometric Evaluation and Instructions for Use.* International Journal of Human-Computer Interaction. 1995.

[5] Fling, Brian. *Mobile Design and Development, 1st Edition.* O'Reilly Media, Inc. August 24, 2009.

[6] gartner.com. *Information technology research and advisory company.* May 19, 2010. *http://www.gartner.com/it/page.jsp?id=1372013* (accessed August 12, 2010).

[7] gartner.com. *Information technology research and advisory company.* February 23, 2010. *http://www.gartner.com/it/page.jsp?id=1306513* (accessed August 08, 2010).

[8] Schiller, Jochen. *Mobile Communications. Second Edition.* Pearson Education. 2003.

[9] Ricci, Francesco. *Mobile Commerce.* 2009. *http://www.inf.unibz.it/ ricci/MS/slides-2009-2010/5-MobileCommerce.pdf* (accessed August 08, 2010).

[10] Ricci, Francesco. *Mobile Recommender Systems.* to be published in Journal of Information Technology and Tourism, 2011.

[11] Herlocker, J. L., Konstan, J. A., Borchers, A., and Riedl, J. *An algorithmic framework for performing collaborative filtering.* In SIGIR '99: Proceedings of the 22nd Annual International ACM SIGIR Conference on Research and Development in Information Retrieval. Berkeley, CA, USA. August 15-19, 1999.

[12] Billsus, D. and Pazzani, M. J. *Adaptive news access. In The Adaptive Web.* Springer Berlin / Heidelberg. 2007.

[13] Bridge, D., Goeker, M., McGinty, L., and Smyth, B. *Case-based recommender systems.* The Knowledge Engineering review. 2006.

[14] Burke, Robin. *Hybrid web recommender systems.* Springer Berlin, Heidelberg. 2007.

[15] Burke, Robin. *Knowledge-based recommender systems.* Encyclopedia of Library and Information Science. 2000.

[16] Hagen, P., with H. Manning and R. Souza. *Smart Personalization.* 1999. *Cambridge, MA: Forrester Research*

[17] Dey, Anind K. *Understanding and using context.* Personal and Ubiquitous Computing. 2001.

[18] Adomavicius, Gediminas and Tuzhilin, Alexander. *Toward the next generation of recommender systems: A survey of the state-of-the-art and possible extensions.* IEEE Transactions on Knowledge and Data Engineering. 2005.

[19] Adomavicius, G., Sankaranarayanan, R., Sen, S. and Tuzhilin, A. *Incorporating contextual information in recommender systems using a multidimensional approach.* ACM Transactins on Information Systems. 2005.

[20] Adomavicius, Gediminas and Tuzhilin, Alexander. *Context-aware recommender systems. Recommender Systems Handbook.* Springer. 2010.

[21] Anand, S. S. and Mobasher, B. *Contextual recommendation. In Lecture Notes In Artificial Intelligence, volume 4737.* Springer. 2007.

[22] cse.iitm.ac.in. *Gradient-Descent Methods.* 2005. *http://www.cse.iitm.ac.in/ cs670/ book/node87.html.* (accessed September 09, 2010).

[23] Kalin, Martin *Java Web Services: Up and Running, 1st Edition.* O'Reilly Media, Inc. 2009.

[24] apple.com. *NSXMLParser Class Reference.* 2010. *http://developer.apple.com /mac/library/documentation/Cocoa/Reference/Foundation/Classes/NSXMLParser_ Class/Reference/Reference.html* (accessed August 05, 2010).

[25] apple.com. *Apple Design Patterns: MVC.* 2010. *http://developer.apple.com /mac/library/DOCUMENTATION/Cocoa/Conceptual/CocoaFundamentals/CocoaDesignPatterns/CocoaDesignPatterns.html* (accessed August 18, 2010).

A. Appendix: Diagrams and Architecture

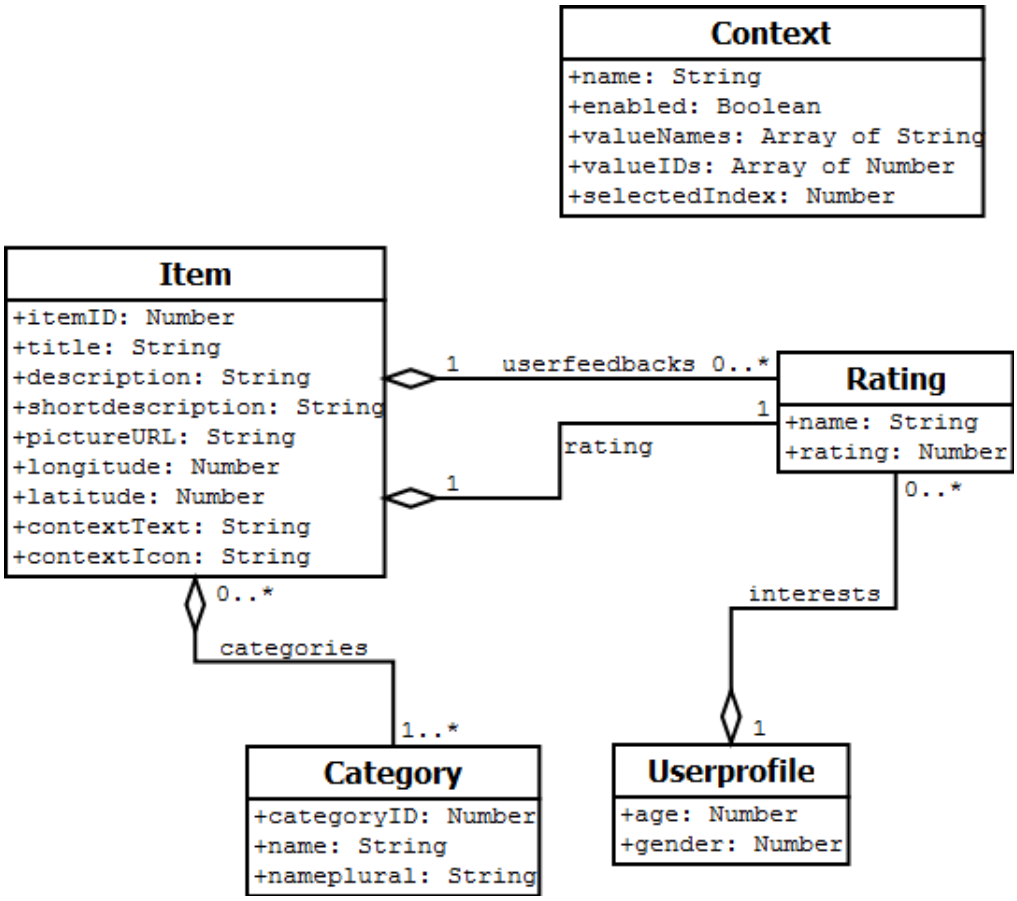

Figure A.1.: Structure of the major part of the Data Model

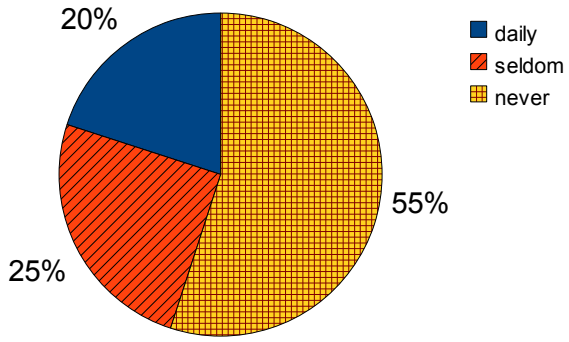

Figure A.2.: The diagram shows the mobile Internet usage of 20 subjects

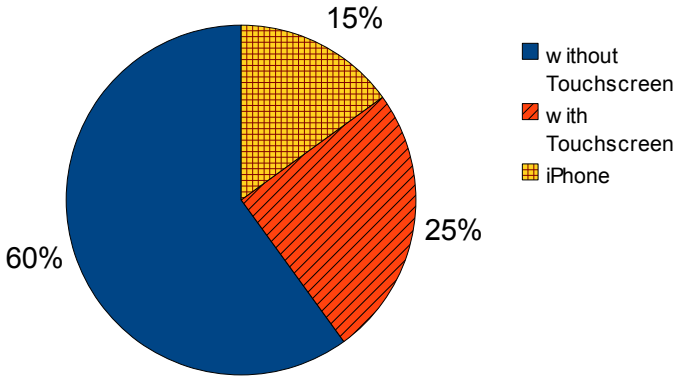

Figure A.3.: The diagram shows the mobile device usage of 20 subjects

B. Appendix: Protocols

B.1. Communication protocol definition: server to client DTD

```
<!ELEMENT rerex (suggestions, wishlist, contextlist, categories
    , contextchange*,itemsForCategory*)>
<!ATTLIST rerex
        version CDATA #REQUIRED
>

<!ELEMENT suggestions (item*)>
<!ELEMENT wishlist (item*)>
<!ELEMENT contextlist (context*)>
<!ELEMENT categories (category*)>
<!ELEMENT contextchange (message)>
<!ATTLIST contextchange
        name CDATA #REQUIRED
        value CDATA #REQUIRED
>
<!ELEMENT itemsForCategory (item*)>
<!ATTLIST itemsForCategory
        id CDATA #REQUIRED
>

<!ELEMENT item (categoryID+, title, description,
    shortdescription, pictureURL, longitude, latitude, rating,
    userfeedback+, contexttext?, change?)>
<!ATTLIST item
        id CDATA #REQUIRED
>
<!ELEMENT categoryID (#PCDATA)>
<!ELEMENT title (#PCDATA)>
<!ELEMENT description (#PCDATA)>
<!ELEMENT shortdescription (#PCDATA)>
<!ELEMENT pictureURL (#PCDATA)>
<!ELEMENT longitude (#PCDATA)>
<!ELEMENT latitude (#PCDATA)>
```

```
<!ELEMENT rating (#PCDATA)>
<!ELEMENT userfeedback (#PCDATA)>
<!ELEMENT contexttext (#PCDATA)>
<!ATTLIST contexttext
        type (high|low) #REQUIRED
>
<!ELEMENT change (reason)>
<!ATTLIST change
        type (add|delete) #REQUIRED
>
<!ELEMENT reason (#PCDATA)>

<!ELEMENT context (contextvalue+)>
<!ATTLIST context
        name CDATA #REQUIRED
>
<!ELEMENT contextvalue (#PCDATA)>
<!ATTLIST contextvalue
        dbid CDATA #REQUIRED
>

<!ELEMENT category (name, nameplural)>
<!ATTLIST category
        id CDATA #REQUIRED
>
<!ELEMENT name (#PCDATA)>
<!ELEMENT nameplural (#PCDATA)>

<!ELEMENT message (#PCDATA)>
```

B.2. Communication protocol definition: client to server DTD

```
<!ELEMENT rerex (user, contextlist, suggestions, wishlist,
    requestItemsForCategory*)>
<!ATTLIST rerex
        version CDATA #REQUIRED
>

<!ELEMENT user (deviceid, age, gender, interests, location)>
<!ELEMENT contextlist (context*)>
<!ELEMENT suggestions (item*)>
<!ELEMENT wishlist (item*)>
<!ELEMENT requestItemsForCategory EMPTY>
<!ATTLIST requestItemsForCategory
        id CDATA #REQUIRED
>

<!ELEMENT deviceid (#PCDATA)>
<!ELEMENT age (#PCDATA)>
<!ELEMENT gender (#PCDATA)>
<!ELEMENT interests (castles, museums, restaurants,
    naturewonders, events)>
<!ELEMENT location (longitude, latitude)>
<!ELEMENT castles (#PCDATA)>
<!ELEMENT museums (#PCDATA)>
<!ELEMENT restaurants (#PCDATA)>
<!ELEMENT naturewonders (#PCDATA)>
<!ELEMENT events (#PCDATA)>
<!ELEMENT longitude (#PCDATA)>
<!ELEMENT latitude (#PCDATA)>

<!ELEMENT context EMPTY>
<!ATTLIST context
     name CDATA #REQUIRED
     value CDATA #REQUIRED
     on (true|false) #REQUIRED
     dbid CDATA #REQUIRED
>

<!ELEMENT item (userfeedback*)>
<!ATTLIST item
        id CDATA #REQUIRED
```

```
>
<!ELEMENT userfeedback (#PCDATA)>
<!ATTLIST item
        name CDATA #REQUIRED
>
```

C. Appendix: Screenshots

Figure C.1.: View for specifying contexts and its values.

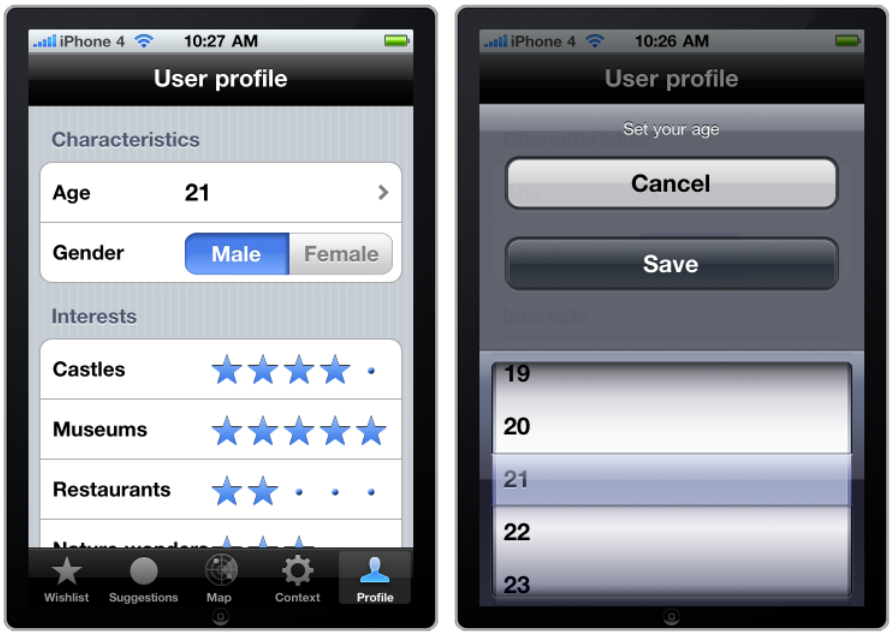

Figure C.2.: View for specifying age, gender and interests.

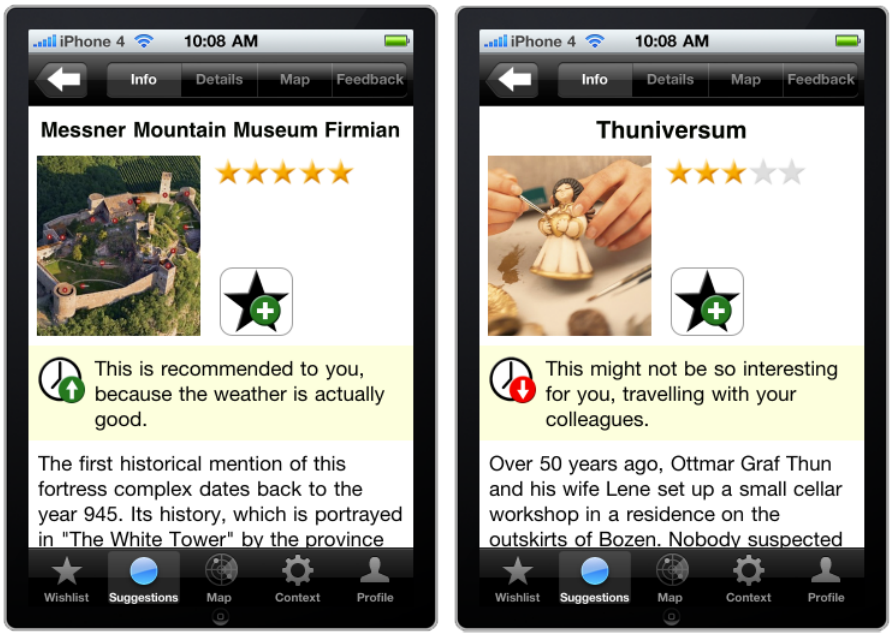

Figure C.3.: Two details views of points of interest, showing positive and a negative contextual influces.

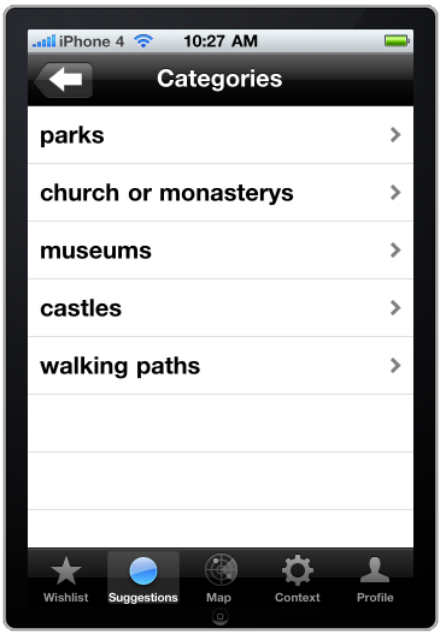

Figure C.4.: Browse through all available items in a category.

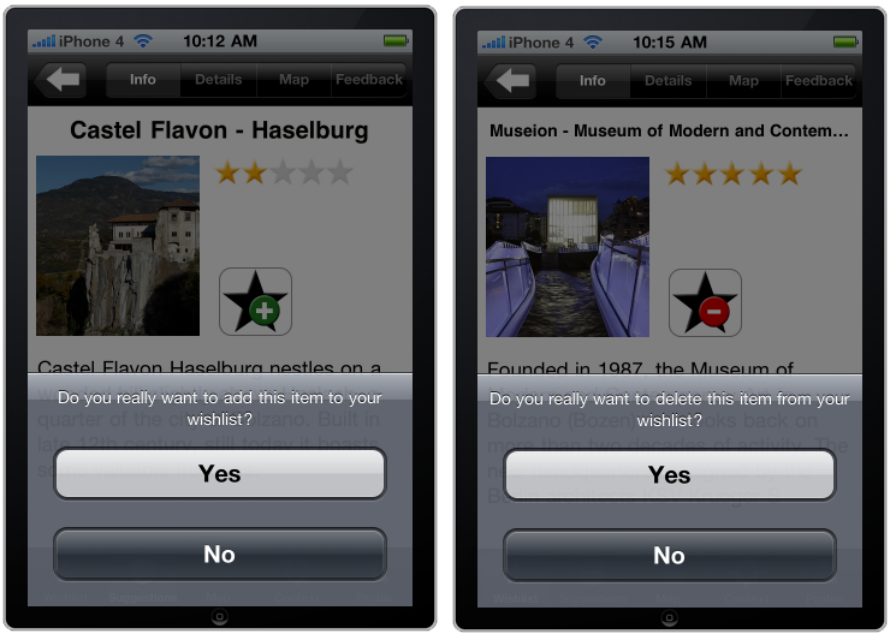

Figure C.5.: Adding and deleting wishlist items.

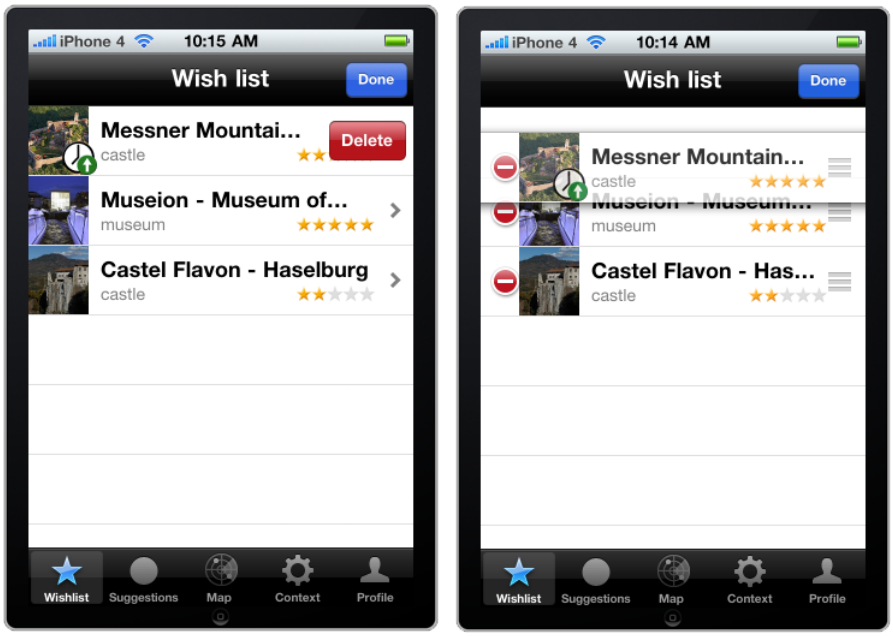

Figure C.6.: Deleting and reordering of wishlist items.

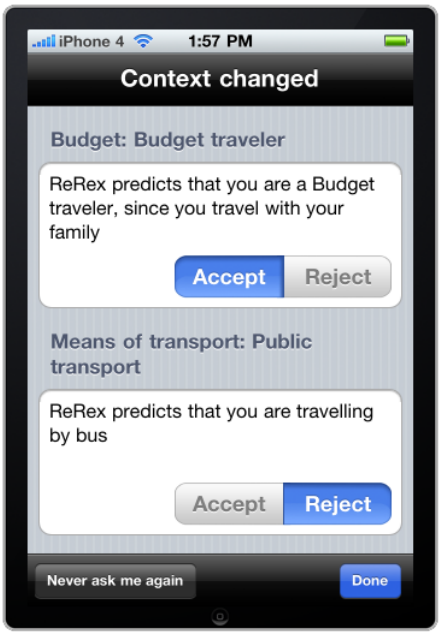

Figure C.7.: Real-Time notification alert for context prediction.

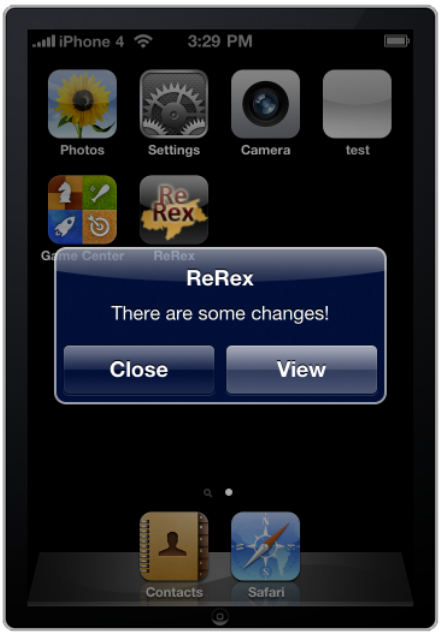

Figure C.8.: Real-Time notification alert if the application is in background.

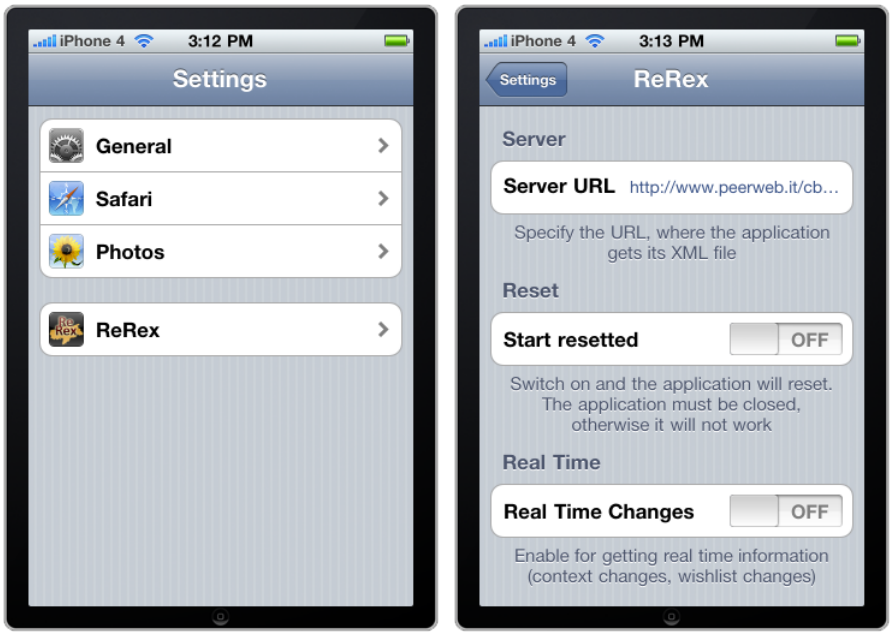

Figure C.9.: There are 3 possible settings for the ReRex application.

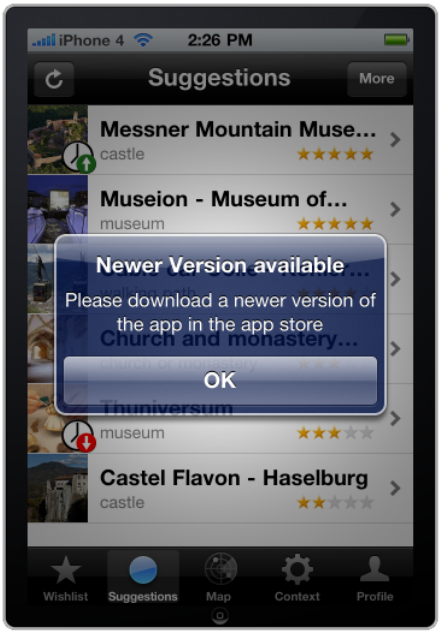

Figure C.10.: Version management of the ReRex application.

D. Appendix: Questionaire

Real-Time Context-Aware
Recommender Systems for Mobile Users

Usage Scenarios

Usage Scenario 1:

Imagine to be a tourist visiting Bolzano.

Suppose that it is a sunny, warm afternoon and you are traveling with your girlfriend/boyfriend. Select a single point of interest of your choice and add it to your wishlist.

Suppose now that starts raining. If needed, revise your previous selection and add another point of interest to your wishlist.

Usage Scenario 2:

Imagine you are living in Bolzano.

Suppose that it is a cold, rainy Wednesday and you are alone. Select a single point of interest of your choice and add it to your wishlist.

Suppose now your parents are coming to visit you. If needed, revise your previous selection and add another point of interest to your wishlist.

Figure D.1.: Two usage scenarios used for the experiments.

Freie Universität Bozen
Libera Università di Bolzano
Free University of Bozen · Bolzano

Real-Time Context-Aware
Recommender Systems for Mobile Users

Questionnaire

On the usability of an iPhone application recommending touristic points of interest.

User ID: _____ **Age:** _____

Start Time: ___ / 09 / 2010 - ___ : ___ **End Time:** ___ : ___

Mobile Internet Usage:

- ☐ Daily
- ☐ Seldom
- ☐ Never

- ☐ Web browsing
- ☐ E-Mail
- ☐ Other applications used: _____

Usual Device:

- ☐ Without touch screen
- ☐ iPhone
- ☐ Other with touch screen

System 1

Please choose the answer, that describes your intention from -2 (strongly disagree) to 2 (strongly agree):

		disagree -2	-1	0	1	2 agree
User interface	It was simple to use this system	☐	☐	☐	☐	☐
	The interface of this system is pleasant	☐	☐	☐	☐	☐
Information	The organization of information provided by the system is clear	☐	☐	☐	☐	☐
	It is easy to find the information I needed	☐	☐	☐	☐	☐
	The system is effective in helping me complete the scenario	☐	☐	☐	☐	☐
Learning	It was easy to learn to use this system	☐	☐	☐	☐	☐
General	Overall, I am satisfied with this system	☐	☐	☐	☐	☐
	I like using this system	☐	☐	☐	☐	☐
	This system has all the functions and capabilities I expect it to have	☐	☐	☐	☐	☐
	Further suggestions: _____					
	_____					
Recommendations	I am satisfied with the suggested points of interest	☐	☐	☐	☐	☐
	I can effectively find interesting suggestions using this system	☐	☐	☐	☐	☐

Comments

Most negative aspects (if any): _____
_____

Most positive aspects (if any): _____
_____

Figure D.2.: Questionnaire part 1. System 1 is the not context-aware version.

System 2

Please choose the answer, that describes your intention from -2 (strongly disagree) to 2 (strongly agree):

		disagree -2 -1 0 1 2 agree
User interface	It was simple to use this system	☐☐☐☐☐
	The interface of this system is pleasant	☐☐☐☐☐
Information	The organization of information provided by the system is clear	☐☐☐☐☐
	It is easy to find the information I needed	☐☐☐☐☐
	The system is effective in helping me complete the scenario	☐☐☐☐☐
Learning	It was easy to learn to use this system	☐☐☐☐☐
General	Overall, I am satisfied with this system	☐☐☐☐☐
	I like using this system	☐☐☐☐☐
	This system has all the functions and capabilities I expect it to have	☐☐☐☐☐
	Further suggestions: _____	
	_____	
Recommendations	I am satisfied with the suggested points of interest	☐☐☐☐☐
	I can effectively find interesting suggestions using this system	☐☐☐☐☐
Context	I understood the benefit of using the contextual conditions	☐☐☐☐☐
	It was easy to specify the desired contextual conditions	☐☐☐☐☐
Contextual Explanations	I am satisfied with the provided contextual explanations	☐☐☐☐☐
	I believe that the contextual explanations are useful	☐☐☐☐☐
	The contextual explanations provided by this system are clear	☐☐☐☐☐
Comments	Most negative aspects (if any): _____	
	_____	
	Most positive aspects (if any): _____	
	_____	

Conclusions

Which system do you prefer?

☐ System 1
☐ System 2

Which system suggested more appropriate points of interest?

☐ System 1
☐ System 2

Thanks for Your Collaboration, Stefan Peer

Thesis Project – Bachelor of Science in Applied Computer Science
University Supervisor: Prof. Dr. Francesco Ricci
University Co- Supervisor: Linas Baltrunas

Figure D.3.: Questionnaire part 2. System 2 is the context-aware version.

E. Appendix: Tables

	Q1	Q2	Q3	Q4	Q5	Q6	Q7	Q8	Q9	Q10	Q11	avg
Mean A	1,45	1,55	1,40	1,40	1,40	1,55	1,30	1,20	1,15	0,80	1,20	1,31
StDev A	0,69	0,60	0,60	0,75	0,68	0,76	0,66	0,89	0,75	1,01	1,01	0,76
Mean B	1,65	1,60	1,40	1,10	0,50	1,50	0,85	0,85	0,70	0,15	0,50	0,98
StDev B	0,49	0,60	0,82	0,97	1,15	0,83	0,99	1,23	1,03	1,09	1,15	0,94

Table E.1.: Means and standard deviations, needed to calculate the p-values for the questions Q1 to Q11, and for the average answer.

	Q14	Q15	Q16	Q17	Q18	avg
Mean	1,30	1,10	1,05	1,50	1,20	1,23
StDev	1,03	0,97	1,05	0,83	0,95	0,97

Table E.2.: Means and standard deviations for the statements Q14 to Q18 (only for system A).

	Q19		Q20	
	System A	System B	System A	System B
Frequency	17	3	19	1
Probable Frequency	10	10	10	10
p-value	0,00174		0,00006	

Table E.3.: Frequency values (obtained and expected) of the answers to the questions Q19 and Q20, expressing the users' preferences about the systems and p-values of the χ^2-Test statistic.

List of Figures

List of Tables