AndroMedia – Towards a Context-aware Mobile Music Recommender

Fredrik Boström

Helsinki May 9, 2008

Master of Science Thesis

UNIVERSITY OF HELSINKI
Department of Computer Science

HELSINGIN YLIOPISTO — HELSINGFORS UNIVERSITET — UNIVERSITY OF HELSINKI

Tiedekunta/Osasto — Fakultet/Sektion — Faculty	Laitos — Institution — Department
Faculty of Science	Department of Computer Science

Tekijä — Författare — Author
Fredrik Boström

Työn nimi — Arbetets titel — Title
AndroMedia – Towards a Context-aware Mobile Music Recommender

Oppiaine — Läroämne — Subject
Computer Science

Työn laji — Arbetets art — Level	Aika — Datum — Month and year	Sivumäärä — Sidoantal — Number of pages
Master of Science Thesis	May 9, 2008	64 pages

Tiivistelmä — Referat — Abstract

Portable music players have made it possible to listen to a personal collection of music in almost every situation, and they are often used during some activity to provide a stimulating audio environment. Studies have demonstrated the effects of music on the human body and mind, indicating that selecting music according to situation can, besides making the situation more enjoyable, also make humans perform better. For example, music can boost performance during physical exercises, alleviate stress and positively affect learning.

We believe that people intuitively select different types of music for different situations. Based on this hypothesis, we propose a portable music player, AndroMedia, designed to provide personalised music recommendations using the user's current context and listening habits together with other user's situational listening patterns. We have developed a prototype that consists of a central server and a PDA client. The client uses Bluetooth sensors to acquire context information and logs user interaction to infer implicit user feedback. The user interface also allows the user to give explicit feedback. Large user interface elements facilitate touch-based usage in busy environments. The prototype provides the necessary framework for using the collected information together with other user's listening history in a context-enhanced collaborative filtering algorithm to generate context-sensitive recommendations. The current implementation is limited to using traditional collaborative filtering algorithms.

We outline the techniques required to create context-aware recommendations and present a survey on mobile context-aware music recommenders found in literature. As opposed to the explored systems, AndroMedia utilises other users' listening habits when suggesting tunes, and does not require any laborious set up processes.

ACM Computing Classification System (CCS):
H.3.1 [Content Analysis and Indexing], H.3.3 [Information Search and Retrieval], H.5.2 [User Interfaces], I.2.6 [Learning], I.5.3 [Clustering]

Avainsanat — Nyckelord — Keywords
Mobile media, context-awareness, recommendation, collaborative filtering

Säilytyspaikka — Förvaringsställe — Where deposited
Kumpula Science Library, serial number C-

Muita tietoja — övriga uppgifter — Additional information

Contents

1 Introduction

Music has played a major role throughout the history of human cultures [WMB01], and is still an integral part of our daily lives. We consume music either actively by listening to the radio or playing records, or passively by being exposed to music in public areas. Music is played in the background of almost any conceivable situation; specifically, a study on the musical experience in everyday life among nearly 350 people indicated that most people were exposed to music on a daily basis and that music was mostly experienced during some activity rather than during deliberate listening [HHN04]. The study also shows that the significance of different qualities in music varies according to location or company, and that the appreciation of music also depends on the company, location and whether the exposure to music was deliberate or not. This strengthens the common conception that music plays a major part in many human activities.

The ubiquitousness of music implies its potential psychological and physiological effects on humans. This assumption has been confirmed by several studies, indicating that music can indeed have many effects on human body and mind. McKinney et al. [MAK+97] found music to have a positive effect on depression, fatigue and stress through a significant decrease in the stress-hormone cortisol. Savan [Sav99] investigated the effect of Mozartean background music on learning among students with special educational needs, and observed an activation in the limbic system of the brain, positively affecting the students' temper through lowered blood pressure, body temperature and pulse rate. Brown et al. [BMP04] conducted an experiment with non-musicians being passively exposed to unfamiliar yet appreciated instrumental music and also observed activation in the limbic system, positively affecting emotion, behaviour and long-term memory.

These effects of music on the human body are bound to have consequences in our daily activities where we are exposed to music either deliberately or by chance. An experiment by Brownley et al. [BMH95] showed that fast, upbeat music has positive effects on training results, especially among untrained runners. Sloboda [Slo91] found that the majority in a group of 83 people had experienced some kind of thrill while listening to music during the last five years. More specifically, 90% had experienced shivers down the spine, 80% lump in the throat, 85% tears and 67% a racing heart. This shows that music not only expresses emotions but also produces emotions [SZ01]. Finally, Brodsky [Bro01] observed the effects of music with different tempo during simulated driving, and noted that the simulated driving

speed was proportional to the tempo of the music. Additionally, traffic violations, like driving against red light, and accidents were more common while the driver was listening to high-tempo music.

These findings indicate that the type of music listened to in different situations may influence the way we perceive the situation, and that the performance of a physical activity can be boosted by being exposed to the right background music. This correlates well with people's tendency to deliberately listen to upbeat music while exercising, peaceful music during romantic occasions and classical music while reading, for example.

The ability to listen to background music in any situation has increased as portable music players like Apple's iPod, Microsoft's Zune, and Nokia's and SonyEricsson's MP3 playing mobile phones become more popular [IFP08]. Still, the choice and organisation of music based on situations can be a time-consuming process. Traditionally, a stream of appreciated music can be obtained by creating playlists and manually adding suitable tunes, or by just playing all tunes at random, skipping tunes when one comes along that is not suitable for the situation. Regardless of the method, getting the right music to play at the right time requires time and effort.

Automatic playlist generation is one solution to the problem. By analysing metadata of music files and grouping tunes, for example by genre, a somewhat homogenous collection can be produced. Metadata-based categorisation assumes, however, that all media files are properly tagged with the right information. This might pose a problem for diverse albums or compilations where the characteristics of each tune may differ, but all tunes are tagged according to the album's general genre. An alternative method is *content-analysis*, which attempts to automatically render a description of an item based on its content. In practice, this approach is only really feasible for textual items since content-analysis methods for audio files still do not calculate tune similarity accurately enough for real world applications [Pam06].

Instead of relying on any properties of the media file itself, an automatic playlist for a certain situation could be generated by keeping track of what the user has listened to previously. Registering listening habits, together with context data, yields a system that after an initial learning phase is able to generate automatic playlists for familiar environments. While a context-aware system would relieve users from managing their music collection, it would still be restricted to the music that is already known. Finding new appealing music is a problem that relates to the increasing supply of digitally available media. To aid customers in the decision process, many online

music suppliers offer recommendation services, the most popular example being Amazon.com [LSY03]. Recommendation systems group users by their taste and calculate which items interest one user based on the preferences of other users. This method of collaboratively sifting out interesting items from the whole collection has become one of the most popular means of finding unknown, yet appealing items. The method is particularly suitable for media items like music, since it does not take into consideration the actual contents of the items.

Recalling the problem of generating playlists for certain situations, we can combine the above methods into a context-aware recommendation system that is able to analyse what music other people have listened to in a particular situation. This is potentially useful in a mobile setting where portable media players are used in varying contexts and where users typically have limited attention resources for managing their music. For example, Patten et al. [PKON04] have shown that the reaction time of car drivers significantly decreases as the cognitive workload caused by using a mobile phone increases. Playlist management or even playback control is thus unfeasible if the user is occupied with other important matters. In this scenario, a context-aware media player could automatically select suitable music even in changing environments.

The challenges in developing context-aware music recommenders are related to collecting and interpreting the user's context and finding listening patterns based on this context. No commercial systems are yet available, and the research prototypes found in literature either do not incorporate enough context data to make them really situation-aware [BBFB$^+$03, JRH06, BJBW07], or deal with recommending music already found on the user's device [WPVS05, RM06, DEO$^+$07, BC08]. To our knowledge, there is no documented attempt to create a mobile context-aware music player with collaborative filtering recommendation techniques.

In this thesis, we present the work done on *AndroMedia*, a mobile context-aware music player prototype with a built-in recommendation service. The context-awareness is based on data from a handful of sensors, both internal and external, providing a good base for situation recognition. The recommendation service is based on collaborative filtering, giving the user a possibility to explore unknown music. We present the architecture and functionality of AndroMedia, as well as discuss design decisions and setbacks in the implementation procedure. The working hypothesis for this work is that people listen to different types of music in different situations. Based on this hypothesis, a user wants, for example, to listen to classical music

while reading a book at home in the sofa, while the playlist would be composed of R&B and hip-hop while jogging in the city. We explore the techniques that realises the scenario mentioned above, namely gathering and interpreting listening habits with accompanying context information and generating recommendations based on context information and other users' listening habits. We also present a survey of mobile context-aware music recommenders found in research literature, listing their strengths and weaknesses, and compare them with AndroMedia. User evaluations of AndroMedia is beyond the scope of this Master's Thesis.

2 Context-awareness

The fundamental idea of context-aware computing is to provide information and/or services to the user based on the user's situation [Dey01]. In order to do that, the application needs to obtain situational data, process it and make use of it in a way that benefits the user. In this section, we discuss what type of context data can be acquired, how it is processed and how applications can benefit from it.

2.1 Data acquisition

An environment, in which a human activity takes place, can be described by a set of features. Each feature describes a dimension in the environment, such as location, temperature, humidity or a list of nearby Bluetooth devices [ASST05]. Each feature has a value that depends on the environment. In a mobile setting, where the user moves about in a changing environment, the values of the features that describe the environment also change over time. Capturing the *context* of an event in time can thus be seen as taking a *snapshot* of the current values of the environment's features at the time when the event occurred.

This context snapshot can be acquired in two ways: explicitly by having the user input the current context or implicitly by using some sensing technology [SBG99]. Explicitly entered context data is common in everyday devices such as mobile phones and PCs. The current time zone, country and other regional settings found in operating systems are all explicitly input context data. The different profiles in a mobile phone are also indicators of context, since users change the profile on their phone according to their environment. Management software of wireless network cards also often have the concept of different locations, such as "home" or "work",

which alter the configuration of the card depending on location. Although humans can accurately describe their context, explicit context data can generally not be considered reliable since the user can be forgetful or simply ignore to supply the application with the right information when the context changes. Also, it is not feasible to require the user to supply the amount and variety of context data needed to produce an accurate understanding of the user's situation.

Capturing the context implicitly using modern technology can provide a rich view of the user's environment. Implicit context acquisition is a matter of reading values from a set of sensors, each of which interprets a certain context dimension. The set of dimensions included in the context view is determined by the available sensors, the diversity of which has lately increased. Some sensors are very common and can be found in almost any handheld device, for example a clock or a battery level monitor. Other, more sophisticated sensors, such as accelerometers, GPS receivers and light sensors, have to be connected or manually integrated into the device in some fashion. In fact, the integration of sophisticated sensors in mobile phones has become more common lately; for instance, Nokia has developed mobile phones with built-in GPS receivers, accelerometers, compasses, thermometers and Near Field Communication (NFC) capabilities. Many modern mobile phones also include an ambient light sensor for controlling the brightness of the display. Furthermore, the built-in microphone can be used as a sensor to pick up ambient noise as context information [KKP+03].

Small, cheap external sensors can be further connected to a device to acquire a more complete view of the user's context. A multitude of sensing devices is available, but only a subset of them provide meaningful data from a context-aware computing perspective, and these can be lumped together into the following families of sensors [May04, Sch02]:

- **Optical data** – properties of light, shapes
- **Audible data** – properties of audio, background noise, voices
- **Motion data** – inclination, direction, movements, actions
- **Location data** – co-ordinates, city, country
- **Orientation data** – magnetic field strength, absolute or relative direction
- **Proximity data** – distance, nearby objects

- **Meteorological data** – temperature, air pressure, humidity
- **Bio data** – heart rate, blood pressure, emotions
- **Identity data** – fingerprints, facial recognition, DNA
- **Other data** – radiation, force, touch

The above families of sensor technologies provide a view of the external conditions and to some extent the state of the user. Another important source of context data is the device itself, excluding any built-in sensors, as it can provide information about the interaction frequency, battery level, GSM signal strength and the application currently executing [May04]. In fact, the device can be seen as a sensor in its own right.

A typical sensor provides data as a discrete time-series of measurements, where the sample rate is specific to the sensor. A snapshot of the context, interpreted by possibly multiple sensors, at any point in time thus results in a sensor data vector

$$[s_1, s_2, ..., s_l]_t \in S_1 \times S_2 \times ... \times S_l \tag{1}$$

where s_i is a sensor value in the dimension S_i at the time t (for example, the value 25 for the dimension *temperature* at time t_5). Although the sensors provide a stream of readings, the application can choose when to store and interpret the data vector. The sample can be stored automatically on certain intervals to acquire an even distribution of measurements. Alternatively, the snapshot can be taken in connection to an event triggered, for example, by the user.

Context data is by nature hierarchical, and can be described with varying granularity, from low level sensor data to aggregated higher level concepts [FW07]. Schmidt et al. [SBG99] propose a hierarchical model for context data with two top nodes and six high level categories, as shown in Table 1. Categories are further divided into features and they into values, which all together constitute four levels of abstraction. Time is also considered to be an own dimension in this model.

The hierarchical property is an important characteristic, since raw sensor data typically needs to be processed and abstracted to some extent before the application can make use of it. The level of abstraction also affects the reliability of implicitly acquired context data, assuming that the probability of erroneous sensor readings is low. Since the combination of raw sensor data into a high level concept, such as

Human factors	User information	Habits, bio-physiological condition, ...
	Social environment	Proximity, social interaction, group dynamics, ...
	Task information	Spontaneity, deliberate tasks, goals, ...
Physical environment	Physical location	Absolute and relative position, proximity, ...
	Infrastructure	Available communication resources, task performance, ...
	Physical conditions	Noise, light, pressure, ...

Table 1: Hierarchical model of context data.

"morning jog", is a result of a series of complex computations, it is only as reliable as the algorithms that infer the context. The high level context can even depend on the user's personal traits, which requires the sensor and/or application to be calibrated to a certain user to provide accurate context information [Sch02].

2.2 Feature extraction

Feature extraction is the task of identifying meaningful information from raw sensor data for further inference. When context is gathered continuously, or at a certain interval, the amount of collected data is typically large and may contain redundancies. For example, a sensor for a contextual dimensions whose value does not change rapidly, such as air pressure, will give the same reading over many samples. A sensor can also provide data that can be expanded into several features. For example, a wireless network card can provide data for features such as a list of access points in range, the number of access points in range, signal strength and transmission rate [MRF03].

The input to the feature extraction phase is a sensor data vector S_t, for some time t, from the acquisition phase. Each dimension of the captured sensor data is processed individually, and the method depends on the type of sensor data being processed. One- or two-dimensional numerical data can be analysed with simple statistical methods like minimum, maximum, mean and median values or standard deviation, while other more complex sensor data needs its own specific method of analysis

[May04]. For example, an accelerometer normally has a higher sampling rate than transmission rate, meaning that a packet transmitted at time t (the snapshot of the dimension) will contain a large amount of actual sensor data. A condensed value for the acceleration feature is acquired using a feature extraction method, such as the mean value, on the data. In contrast, the information gathered from a Bluetooth transceiver consists, for example, of a set of MAC addresses and has to be analysed in a domain-specific way in order to acquire relevant features. This could be done by counting distinct MAC addresses.

After extracting meaningful features, the process generates a *feature vector*

$$[f_1, f_2, ..., f_m]_t \in F_1, F_2, ..., F_m \tag{2}$$

where f_i is a feature sample (e.g., "5") in the feature dimension F_i (e.g., "number of nearby Bluetooth devices") for the time t defined by the sensor vector S_t. The number of dimensions in the resulting *feature space* is not necessarily equal to the number of raw data dimensions, since sensor data can be split into more or combined into fewer feature dimensions than sensor dimensions.

2.3 Context comparison

Since context-awareness is based on identifying situations familiar to the user, context-aware applications need to be able to compare different context snapshots and determine their similarity. There are at least two approaches to comparing contexts, and the ones presented here are both based on the feature vectors produced by the feature extraction process.

If the application requires the context data to consist of numerical data, the feature values need to be converted into integers or floating point numbers. This is trivial for features based on numerical sensor data, but if the feature space contains list-based features or non-numeric features, such as nearby Bluetooth MAC addresses or the ESSID of wireless networks, the data needs to be converted into numerical values. This can be done using, for example, some transformation into binary values where one bit corresponds to a possible value [May04]. However, such a transformation is unfeasible when the set of possible values grows.

An alternative to the above strategy is to define a *distance metric* on each feature dimension [May04]. Formally, the distance between two samples of a feature F is

calculated using the similarity between the samples:

$$d : F \times F \rightarrow [0 : 1] \tag{3}$$

The distance measure yields a normalised value between 0 and 1.

By defining this operation on each type of feature, the feature space gets the required interface for acting as input to different comparison algorithms, such as classification or clustering. For numerical values, the distance operation is trivial, but for complex features, it has to be defined separately for each feature. For example, the similarity between two samples of the feature dimension "nearby Bluetooth devices", where a sample consists of a set of MAC addresses, could be calculated using the Hamming distance

$$d(f_1, f_2) := (f_1 \setminus f_2) \cup (f_2 \setminus f_1) \tag{4}$$

where f_1 and f_2 are sets of MAC addresses. The Hamming distance essentially counts the number of differing elements in two sets.

By combining this method of calculating context similarity with the personalisation techniques described in the next section, we can produce recommendations based on the user's situation, as described in Section 3.6.2.

3 Recommending Music

Since the mid 90's, traditional recommendation systems have been used in commercial applications to recommend books [LSY03], news articles [RIS+94] and web sites[1] to potential customers and users. Although most of the techniques used in traditional recommendation systems can also be used for recommending music, there are some specific aspects involved in recommending digital media. In the following subsections, we will give an introduction to the problems and solutions surrounding recommendation systems, especially when recommending music to listeners. We will also explore how context information can be used in co-operation with traditional recommendation systems to create context-sensitive recommenders.

[1] http://www.stumbleupon.com

3.1 The Problem

With the steady increase in the availability of digital content, an increasing difficulty in finding the things that actually meet our needs emerges. Modern information retrieval (IR) systems provide very good means of searching for practically anything on the web using a few keywords. IR applications, however, are not aware of the personal preferences and interests of the user, and thus treat all users in the same manner. A search for "Mozart" would yield the same results, regardless if the user were a teenager looking for piano notes or a Ph.D. student writing a thesis on classical 18th century music. The search result for such a broad topic on the Google search engine returns over 35 million documents[2], the ten most highly rated of which include information about the composer W. A. Mozart, a coffee roaster in Texas, USA and a piece of music notation software.

A search for a very specific subject could similarly yield a result with mostly irrelevant documents due to subject rarity, leading to a similar problem of searching among the search results. Dealing with this information overload is usually a very time consuming process, which could be alleviated using some additional aids. For example, the retrieval application could possess some kind of knowledge about the users' identities and the kind of information they seek without them having to instruct the system each time they are looking for additional information on a previously researched subject. Along the line, the application could even become proactive and tell users what they are looking for before they know it themselves.

The traditional way of coping with information overload is using our social contacts [SM95]. People have a tendency to ask their family, friends and colleagues for hints when having to make a decision between a set of items without sufficient knowledge. Furthermore, over time we learn to trust certain people for certain subject areas, as we gain knowledge about their personal expertise and get experiences from their previous suggestions. Consider the following example:

> During a coffee break, Stacy discusses the new Robbie Williams album with her colleagues. She already owns a few of Robbie's CDs and enjoys his music, but isn't sure about this one. James, formerly unfamiliar with Robbie's music, had already bought the album and liked it very much. Both Sarah and John, however, had heard a few tunes and were not too fond of the new CD, although they had liked Robbie's previous

[2]http://www.google.fi/search?q=mozart, retrieved 2008-03-31

> material. Since Sarah knows that James does not share her music taste, while Sarah and John are Robbie Williams fans just as herself, she trusts her like-minded friends' judgement and does not buy the new CD.

This process has a number of shortcomings. First, the diversity of opinions a person gets depends on the number of friends she knows and interacts with. If only consulting our closest work colleagues or family members, one would not get a recommendation as exact and personalised as when consulting a larger group of people. Second, recommendations have in general to be asked for. We have to initiate a dialogue or otherwise come in contact with our social network in order to receive suggestions and learn from their opinions. This can take time and requires active searching for recommendations.

These shortcomings can be overcome by means of information technology. A number of different recommendation techniques have been developed to automate the process of satisfying the user's information needs. The fundamental idea is to develop an understanding of the user's needs and look up available items matching this information. In other words, recommendation systems can be seen as filters that allow interesting items to pass while sifting out unwanted items.

The first implementations of such filtering systems date back to the early 1990s [GNOT92, HSRF95, SM95]. At that time, the amount of electronic mail sent and received had already reached a fairly high level, causing e-mail flooding. With the rapid growth of the World Wide Web, information overload became an increasing problem in the 90's. After nearly two decades of research in the area, two main approaches make up the status quo of recommendation systems: content-based filtering and collaborative filtering. Additionally, combinations and modifications of these two have been developed for greater accuracy, efficiency and customisation [AT05].

Content-based filtering can be seen as an extension of traditional information retrieval systems in the sense that it uses a description (automatically derived or manually annotated) of available items as basis for its recommendations. Together with the user's usage history, the system tries to find common features among the items the user has found interesting in the past and tries to predict how well an unseen item would match the user's information need [BHC98, AT05].

Collaborative filtering, on the other hand, is not concerned with the contents of the items, but is based on the idea that people who have agreed in the past, tend to also

agree in the future [RIS+94]. The information about how different users have rated the same items is used to find people with similar preferences. Recommendable items are those unseen items that other similar users have given a high rating. A rating in this context is either an explicit user rating or an implicit rating inferred from the user's actions.

When dealing with digital media, such as music and video, some additional difficulties have to be taken into consideration in the recommendation process. Humans can sense many different aspects of music, for example, tempo, beat, melody, lyrics and instruments, but it is computationally difficult to compare two tunes in the same way a human does. Additionally, when recommending music for listening, as opposed to buying, the user probably wants a popular item to be recommended more than once. These challenges can be tackled, although not with human accuracy, with contemporary recommendation systems.

Although the available recommendation methods might be based on quite different ideas, they all serve a single purpose: to get the right information to the user with minimal effort. This implies a series of independent sub-actions. First, we have to identify which information the user is interested in and find a way to obtain this information. Next, we need to generate recommendations using this information with one of the aforementioned techniques. Last, we should continuously update our knowledge about the user to be constantly able to recommend relevant items, even when the user's interests changes over time.

In subsequent sections, we will examine these tasks involved in recommending items to users, especially focusing on the challenge in recommending music, and explain how we can benefit from these technologies.

3.2 User Modelling

As noted earlier, recommendation systems aim at getting the right information users with minimal effort. Minimal effort in this context means that users should not have to rephrase their interests and information needs to the system each time they use it. Instead, the application should autonomically learn about the users, what they are interested in, what they already know, and what information each user would want to receive. This is achieved by developing a *user profile*, or *user model*, internal to the system and specific for each user. The user model formalises the user's preferences and constitutes the basis for recommendations made to the user. In both content-

	Robbie	Katie	Corinne	Norah
Stacy	5	2		3

Table 2: Example of part of a user model in collaborative filtering

based and collaborative filtering, the user model consists of parameters used by the recommendation algorithms, for example user-item ratings, demographic data or similarity measures. In content-based filtering, the user's model is used to find items similar in content to those the user has liked before. In collaborative filtering, the model would be used to find like-minded users and retrieve unseen items that these users have rated highly. Simply put, content-based filtering does item analysis and comparison, while collaborative filtering compares users to each other, and both methods use the model of the user's preferences as a basis for recommendations.

In its simplest form, the model is either a collection of weighted keywords describing the user's topic of interest, or numerical ratings for seen items. In content-based filtering, the model generally contains a subset of all keywords that describe the items interesting to the user. The keywords are weighted according to occurrence to indicate how well the keyword describes the user's overall interest. Techniques for calculating user specific keyword vectors include the Rocchio algorithm, Bayesian classifiers and the Winnow algorithm [AT05]. In collaborative filtering, on the other hand, the user model consists of a vector of ratings for all items in the collection. If the user has not rated an item, or if the item is unknown to the user, no rating is stored for that item.

For example, suppose that Stacy likes Robbie Williams, Norah Jones and Katie Melua. Her user profile in a content-based filtering system for music might include keywords like "jazz", "pop", "vocal" and "rock". In a collaborative filtering system recommending artists, she might have rated Robbie higher than Katie and Norah, while still being unfamiliar with Corinne Bailey Rae. A subset of her user profile could then be described by Table 2.

The information used to build the user profile can be obtained explicitly by asking users to supply information about their preferences to the system, or it can be collected implicitly by analysing each user's interactions with the system over time. The former method could be implemented as a questionnaire that users have to answer before starting to use the system, or by requiring users to rate a set of items in order to obtain an initial set of items the user is interested in. While this approach is more straightforward and instantly yields a model that is ready for

use, it requires all users to manually input information into the system and thus is more obtrusive. Unless there is a clear benefit for the user from supplying personal preferences and giving continuous explicit feedback, such information would likely not be provided, making it impossible to build a reliable user model [Nic98]. A model based exclusively on explicit feedback also has to be revised regularly. The user model will be outdated if users' interests change over time, but they fail to inform the system about the change [GSCM07].

Implicitly collected data, on the other hand, is obtained by observing how the system is being used and *inferring* the user's interests. This method is completely unobtrusive and allows the user to immediately start using the application. The user model is not, however, accurate until the user has interacted with the system for a certain amount of time. Building an accurate user model from implicit feedback is also more difficult since it is not always clear when the feedback is positive or negative. For example, if a user skips a tune in the current playlist, the tune is probably disliked, but if the user listens to the tune in its full length there is no way of knowing if it is a favourite or if it is found only mediocre. Thus, we can only make qualified guesses whether the implicit feedback is positive, neutral or negative.

Although the feedback from an implicit approach can be considered less valuable and reliable, it is superior in amount since implicit feedback is collected whether users explicitly indicate their opinion or not. This suggests a trade-off between the cost of collecting feedback and the value it produces [Nic98]. To receive the highest benefit, the two approaches can be combined into a hybrid feedback scheme. For example, the user could manually supply one keyword upon which the system builds a crude user model, refining it over time using either explicit or implicit feedback from the user. Another approach would be to use implicit feedback to reinforce explicit feedback.

3.3 Content-based Filtering

Content-based filtering originates from the area of information retrieval, where the focus originally was on matching records to information requests, and to retrieve the matching records for the user [Sal89]. Although the characteristics of information retrieval and content-based filtering seem somewhat similar, there are a few fundamental differences [BC92], the most important of which is the way they accept user input. An information retrieval system accepts a textual query written by the user, while content-filtering system accepts a model of the user's interest. Since the

model is built and reinforced over time, filtering systems also generally yield results matching a long term interest, while retrieval systems typically do not use search history to improve their search results.

These key differences characterise typical content-based filtering systems, as they are designed to help the user find more relevant items by learning the user's interest over time. The learnt profile of interests is used as a more accurate substitute for keywords in the retrieval process, and makes up half of the comparison procedure. The other half is the collection of recommendable items, whose contents have to be analysed and formalised to fit the comparison algorithm. The comparison itself produces a score for each item denoting how well it corresponds to the user's interest. The items can be displayed to the user in a ranked list, where items at the top of the list are more likely to be of interest to the user. The user can typically give feedback to the system on how well it predicted the interesting items by either giving positive or negative feedback to single recommended items. Using this information, the system learns about the user and can refine the profile for more accurate recommendations in the future.

3.3.1 Content Analysis

The process of formalising an item into a set of keywords is traditionally called *indexing*, and originates from the information retrieval area. Next, we outline the traditional indexing method, which was originally designed for text documents. We also discuss how this method can be applied on music and investigate some dedicated content analysis methods for media files.

Intuitively, an arbitrary document in a collection of text documents could be found by scanning each document for certain keywords and returning the documents that contain most of these keywords. This method quickly becomes too expensive and time consuming as the documents and document collection grow in size [SM86]. Instead, filtering and retrieval systems use a document profile to represent the contents of the document. The profile consists of words that accurately discriminate the document from the rest of the collection. These *index terms* are generally words that occur often in the indexed document, but seldom in other documents in the collection, calculated, for example, with the widely used $tf \cdot idf$ algorithm [Sal89].

Using the $tf \cdot idf$ algorithm to find good index terms for a document, we first have to remove all *stop words*, i.e., words that occur frequently in all documents in the

collection. Examples of stop words in English include the words "and", "the", "of" and "to". Next, we calculate the term frequency tf_{ij} for the rest of the terms T_j in every document D_i in the collection. The term frequency describes how many times the term T_j occurs in the document D_i and is a candidate for selecting index terms for a document. However, the fact that a term occurs frequently in a document does not guarantee that the term discriminates the document from the rest of the collection, since a term with a high frequency in one document also can occur frequently in other documents in the collection. Consider, for example, a document in a collection of medical journals. After analysing the document, words like "fever", "patient" and "syndrome" got the highest term frequency values. However, since most other documents in the collection also contain high amounts of these words, a keyword search using these terms would not rank the analysed document higher compared to the rest of the collection.

In addition to terms that occur often in a document, we should thus be looking for terms that accurately identify the document in the whole collection. These are terms that occur frequently in one document, but are rare in other documents. For this measure, a common approach is to use the inverse of the document term frequency df_j, which denotes the number of documents in the collection where term T_j occurs. The resulting inverted document frequency idf_j, calculated as $\log(N/df_j)$, where N is the collection size, is considered a trustworthy measure of the ability of the term T_j to discriminate the document D_i from the rest of the collection of N documents [Sal89].

Combining these two measures, the term frequency tf_{ij} and the inverted document frequency idf_j, we obtain a weight for each term in the document that describes the word's suitability as an identifying term for the document. The weight is called the $tf \cdot idf$ score of a document and is calculated using

$$w_{ij} = tf_{ij} \cdot \log \frac{N}{df_j}. \tag{5}$$

Finally, we select all terms with a sufficiently high $tf \cdot idf$ weight as the index terms for the document.

This approach works well for textual documents, but cannot be directly applied on digital media, such as music. Obtaining a reliable formalised representation of music files is not an easy task. The $tf \cdot idf$ measure can be used on metadata such as free text comments, but these are normally rather small if even provided. Song metadata like author, title, genre, album etc. can be directly used as keywords, if all tunes are

properly tagged [PB07]. Some systems also employ a group of experts who listen to and annotate tunes manually [Pam06]. While this approach results in a reliable tune description since it is made by humans, it is an expensive and slow process since it is done manually. Another source for descriptions of music files is the Internet. Various data about a tune, album or artist can be obtained from online databases, such as Gracenote[3] or MusicBrainz[4], using some kind of digital fingerprint of the audio medium. Textual descriptions of albums or artists can be analysed using the text analysis method described above. Since on-line databases can be maintained by an active community of ordinary users, the amount of tunes covered is potentially much greater than in the case of experts doing the job [Pam06]. On the other hand, the data obtained from user maintained sources is often not as reliable and diverse as if a dedicated group of people was doing the annotation.

A fair amount of research has also been done in the computational analysis of the actual audio signal of the music file [Dow06]. This method is both cheap and fast but does not produce as flavoured descriptions as when humans annotate music. While high level features, such as the genre or general mood of a tune, would perhaps be beneficial in a recommendation context, the best performing analysis algorithms have an accuracy of only around 65% [Mir07]. Furthermore, the algorithms available today cannot determine the audio quality, making a garage band tape no different from a concert hall recording. They can, however, analyse low level characteristics that can be used to describe certain aspects of the audio file and compare it to others [Pam06].

3.3.2 Calculating Similarity and Generating Recommendations

In the following section, we are only considering the case where both items in the collection and the user profile have been formalised into weighted keyword vectors using the methods described in earlier sections. The core task of a recommendation system is to help users find those items of the collection that most closely matches their individual interest. This is accomplished by comparing the similarity between the user model and all available items, sorting the items according to their similarity score and returning the sorted collection to the user. An alternative to avoid too big results is to only return to the user a set of highest ranked items. There is a range of different comparison algorithms for different purposes, but the perhaps most widely

[3] http://www.gracenote.com
[4] http://www.musicbrainz.org

used matching technique is the vector space model [AT05].

In the vector space model, each item is seen as a vector in a multi-dimensional vector space [SM86]. The elements of the vector are the weighted index terms describing that item. A user model is also considered a vector in the same space, and the similarity between an item and the model is calculated as the "closeness" of the vectors in the vector space. Usually, the closeness is calculated using the cosine similarity measure [AT05, Sal89]:

$$\cos(w_d, w_u) = \frac{w_d \cdot w_u}{||w_d|| \cdot ||w_u||} = \frac{\sum_{i=1}^{K} w_{i,d} w_{i,u}}{\sqrt{\sum_{i=1}^{K} w_{i,d}^2} \sqrt{\sum_{i=1}^{K} w_{i,u}^2}}, \tag{6}$$

where K is the total number of index terms in the collection, w_d is the item keyword vector, w_u is the user model vector, $w_{i,d}$ is the ith term in the item keyword vector and $w_{i,u}$ is the ith term in the user model vector.

The cosine similarity is considered a reliable measurement of how well an item corresponds to an interest of the user. In the text case of text documents, for example, a user interested in baroque music has probably viewed documents that assign high weights to keywords related to that topic. The user modelling techniques then assign to the user's profile keywords from these documents that have high weights. Similarly, unseen documents about baroque music will have high weights for the same keywords, which yields a high similarity value [AT05].

3.3.3 Strengths and Weaknesses

Content-based filtering techniques have some well-known drawbacks [BS97]. Since both user models and document vectors are based on keywords derived from the items in the collection, they are restricted to the performance of the extraction method used to analyse the items. The resulting recommendations are thus only as accurate as the keywords describing the documents. This is particularly serious with items whose content cannot be reliably analysed automatically, like music or video. In the case of documents, contemporary content-analysis algorithms also do not take visual aspects of the item into consideration, such as layout or occurrence of images, although these may affect how a human perceives the item. This problem is often referred to as the *limited content analysis* problem [AT05].

The second shortcoming is *over-specialisation*, i.e., that content-based systems can recommend only items that are similar to the user's profile. When the user views

these items, the corresponding keywords in the user profile are strengthened, resulting in more similar items being recommended to the user. Over time, the user model will be specialised on a narrow area, not allowing differing documents to be recommended. This problem is traditionally overcome by infusing some randomness into the recommending algorithm [BS97].

A third problem is that the system needs some initial understanding of new users in order to give any kind of recommendations to them. The system can develop a reliable user profile only after a user has interacted with the system for some period of time. This *new user problem* is usually solved by letting the user select some initial set of appreciated items before starting to use the system. If the initial user profile contains demographic data like age or location, this could also be used to select some initial items that would likely interest the user. Another alternative is to initially recommend some items that are considered popular among the rest of the users [SFHS07].

On the other hand, since content-based filtering uses item contents as basis for recommendations, even newly added and less popular items can be recommended. This can result in good recommendations for a user who is interested in a very specialised niche. As we discuss next, this is not the case with collaborative filtering, where recommendations are dependent on other users' ratings.

3.4 Collaborative Filtering

As opposed to content-based methods, which only take into consideration one user and interests of the user, collaborative filtering is based on the item ratings by users in a collaborative community. The idea is that users whose ratings correlate have similar tastes and are likely to share opinions about unseen items. One benefit of collaborative filtering is that the contents of the items are not taken into consideration, which means that the recommendation system will perform equally well regardless of item type.

The basis for collaborative filtering systems is a user model that consists of the user's ratings for items in the collection. As noted in Section 3.2, these ratings can be obtained either explicitly or implicitly or by a combination of these. A set of different user models can be visualised in a user-item matrix, see Table 3. The matrix consists of the users in the system and their individual ratings for the items in the collection. A missing value means that the user has not seen or rated the

	Elton John	Shakira	Aretha Franklin	Franz Schubert
Sarah	5		4	1
John	4	2	5	
Stacy	1	3	2	5
James	2	1	4	5

Table 3: A user-item matrix. In our example, the items correspond to artists.

item.

3.4.1 Collaborative Filtering Algorithms

The available collaborative filtering algorithms can be categorised into memory-based and model-based algorithms [BHK98]. Memory-based algorithms require that the whole collection of users' ratings is brought into memory and scanned every time a score for a new item is calculated. This class of algorithms is user-centric in that they try to find other similar users, *neighbours*, to the active user. The prediction for an unseen item is calculated using the neighbours' ratings for that item. For example, given an active user u and a set of neighbours $v \in N$ to that user, we can predict the rating the user u would give the unseen item i as

$$pred(u,i) = \bar{r}_u + \frac{\sum_{v \in N} sim(u,v) \cdot (r_{vi} - \bar{r}_v)}{\sum_{v \in N} sim(u,v)}, \tag{7}$$

where r_{vi} denotes the rating by neighbour v on item i, and $\bar{r}_u$ and $\bar{r}_v$ are the mean values of all ratings given by the user u and the neighbour v, respectively . Equation 7 is an extension of a naive average-rating algorithm that only calculates the mean rating for item i by all neighbours $v \in N$. This extended algorithm takes into account that some neighbours are more similar to the user than others by including a normalised similarity weight $sim(u,v)$, measuring how similar the user u is to the neighbour v and weighing the impact of that neighbour's ratings accordingly. Furthermore, it takes into consideration that some neighbours only use the low or high end of the scale in their ratings. This is corrected by adjusting each neighbour's ratings according to that neighbour's mean rating ($r_{vi} - \bar{r}_v$ in Equation 7) [SFHS07].

The similarity calculation $sim(u,v)$ between the user and a neighbour is generally based on *co-rated* items, which both the user and a neighbour have rated [AT05, SFHS07]. A popular approach for calculating similarity between users is the Pearson

correlation coefficient:

$$sim(u,v) = \frac{\sum_{i \in CR_{u,v}} (r_{ui} - \bar{r}_u)(r_{vi} - \bar{r}_v)}{\sqrt{\sum_{i \in CR_{u,v}} (r_{ui} - \bar{r}_u)^2} \sqrt{\sum_{i \in CR_{u,v}} (r_{vi} - \bar{r}_v)^2}}, \tag{8}$$

where $CR_{u,v}$ denotes the set of co-rated items between user u and neighbour v [SM95]. The Pearson correlation coefficient yields a value between -1.0 and 1.0. Other similarity measures include the *cosine similarity* [BHK98] used, for example, in content-based filtering to calculate similarity between users' interests and items (see Equation 6 in Section 3.3.2).

While quite intuitive and straightforward, the space and time requirements of the memory-based algorithms are linearly proportional to the number of ratings. This makes the approach unusable when the size of the collection grows.

Model-based algorithms, on the other hand, use the data set to create an underlying model that is used to predict ratings. They have a more probabilistic approach as they try to calculate the probability that the user would rate an item a certain way given their previous ratings. Assuming that ratings have an integer value from 0 to m, the expected rating r for item i by user u is

$$E(r_{ui}) = \sum_{q=0}^{m} q \cdot \Pr\left(r_{ui} = q | r_{uj}, j \in I_u\right) \tag{9}$$

where the probability expression is the probability that user u gives a rating with value q to item i given the user's previous voting for items j in the set I_u of items that the user u has rated [BHK98].

The probability can be calculated in many ways, of which a Bayesian network approach is among the most popular [SFHS07]. In this model, the idea is to develop a probability model denoting how item ratings depend on each other, and use this model as a basis for calculating the probability for a certain rating of a non-rated item. This is done using the collection of items and the user's previous ratings of those items, including non-rated items, to create a Bayesian network where each node corresponds to an item in the collection. The state of a node corresponds to the different rating values the item can receive, and an additional state for a missing rating. The parents of a node are then the best items to use for predicting the item's rating [CHM97].

3.4.2 Strengths and Weaknesses

Collaborative filtering overcomes some of the problems with content-based filtering. As noted, recommendations are made solely on other users' opinions, which means that collaborative recommenders can recommend any type of item regardless of content type. This is perhaps the most important benefit of collaborative filtering systems when recommending music.

However, also the collaborative filtering approach to recommending items has its shortcomings. The new user problem occurring in content-based systems is also present in this approach, see Section 3.3.3. *Sparsity* is another problem related to the relative amount of users, items and ratings. If the number of items is much greater than the number of users providing ratings, many items will have quite few ratings. This results in low visibility for those items, although the ratings were high. In general, this also means that only popular items are recommended, which can lead to users with a narrow field of interest not getting accurate recommendations for that area [AT05].

There is also a problem with the visibility of new items, called the *cold-start problem*, something that content-based filtering does not have. Since recommendations are based on what other people have thought of an item, that item has to have some ratings in order to be recommended. Items recently introduced to the system will naturally not have any ratings, and can thus not be recommended. This problem can also be overcome by introducing an element of randomness to the recommendations, or by combining content-based and collaborative methods [SPUP02].

3.5 Hybrid Recommenders

Although content-based filtering and collaborative filtering are regarded the most used approaches for recommending items, there are a handful of other techniques. *Demographic filtering* is based on users' personal attributes, such as age and gender, and generates recommendations based on demographical classes. *Knowledge-based recommenders* have knowledge about how a particular item meets a particular user need and uses that knowledge to reason about needs and possible recommendations. Last, *utility-based recommenders* suggest items based on calculations on how useful an item would be for a particular user [Bur02].

Potentially, all recommendation techniques can be combined in different ways since many of the issues with either approach are overcome by characteristics in the other.

The combination of different recommendation approaches into *hybrid systems* uses qualities from the combined methods to boost performance and minimise the drawbacks of each system alone [BS97]. The different recommendation methods can be combined on different levels: from using the output of one technique as input to another to combining properties from respective sources to form a combined algorithm. Other methods of combining include mixing different techniques or using one or the other depending on the situation. The most common recommendation systems to be combined are collaborative filtering and content-based filtering, and the reason for combining the two methods is to overcome the new user and cold-start problems [Bur02].

Although many of the possible combinations have not been explored, evaluations of existing hybrid systems combining collaborative filtering and content-based filtering indicate that hybrid systems outperform the two approaches in recommendation accuracy compared to when they are used alone [YGK+06].

3.6 Context-based Filtering

When considering mobile applications, an interesting aspect of the recommendation problem is in what environment the items are used and recommendations are made. Users may behave differently and have different information needs in different situations, but traditional recommendation systems do not take this into consideration. Taking into account contextual information like time, location or company, the user may receive recommendations with greater accuracy than she otherwise would.

For example, consider a mobile music player that realises that it is Thursday morning and the device is being used in a car driving towards a location tagged as "work". By logging previous usage events, it also knows that the user normally listens to classical music while driving to work. If the system furthermore had access to other users' preferences, the recommendation mechanism could suggest tunes that other people with the same musical taste had liked in the same situation. If the device could connect to a music repository and download tunes or previews of tunes, it could even generate a playlist of formerly unknown music that would fit the user's taste in this particular situation. This scenario could be achieved by incorporating contextual data into conventional recommendation systems.

Recalling the hierarchy of context data by Schmidt presented in Section 2.1, which consisted of two top nodes, *human factors* and *physical environment*, we note that

the former category, consisting of the user's habits, social contacts etc., is precisely the information that a traditional collaborative filtering system bases its recommendations on. By adding the other branch of the hierarchy, consisting of location, temperature, time etc., we obtain a context-aware collaborative filtering system [Che05]. This observation reinforces the motivation to exploring context-aware recommendation systems with collaborative filtering as a foundation.

Incorporating context data into a collaborative filtering system is a matter of extending the traditional user-item ratings with information about the environment in which the rating was given. In other words, the system needs to take a *snapshot* of the current context when a rating is made [Che05]. This might pose a problem for systems where items can be rated *after* the user has experienced it, for example when rating a movie after coming home from the cinema. To avoid this, the recommendation system can rely on implicit user feedback, registering and interpreting the user's interactions together with the current context while the user is enjoying the experience. For example, a mobile music recommender could give a positive rating to a tune when the user has listened to it for more than half of its duration, registering the current context at the same time.

Next we will explore two different approaches to generating recommendations, given that the system contains user-item ratings and associated context data.

3.6.1 Dimension Reduction

A traditional recommendation system can be seen as using a two-dimensional space of values as base for its recommendations. In the case of collaborative filtering, this space would consist of user-item ratings. By adding contextual information to recommendation systems, the two-dimensional space is extended with further dimensions, such as time or location, into a multi-dimensional recommendation space. The same multi-dimensional model is used for example in data warehousing and Online Analytical Processing (OLAP) applications [ASST05].

Using this model, the multidimensional recommendation problem can be transformed into a two-dimensional user-item recommendation problem by reducing the dimensions. Consider a traditional recommendations system that, for example, recommends tunes to users using one of the existing two-dimensional approaches discussed earlier, in this example collaborative filtering. Suppose that we add a dimension *time* to create a time-sensitive recommendation space with three dimensions:

users, items and time. Depending on the distributions of user-item ratings in the time dimension, the third dimension can potentially be used to generate context-based recommendations. We now have two possibilities:

1. If there is no visible trend in rating values in different segments of the time dimension, we cannot infer that some tunes are more popular during certain periods of time and thus cannot recommend any tunes based on the time dimension.

2. If there is a clear correlation between popularity of certain tunes during certain periods of time, we can use the time dimension to calculate recommendations using the time dimension.

To exemplify the above observations, consider the following scenario. Sarah is using a music recommendation application and has rated tunes at different times of the day and in the locations "home" and "work". If, when analysing the data, it becomes clear that Sarah's ratings are similar in both "home" and "work" contexts, the location dimension cannot be used as a discriminator to distinguish what type of music he likes better in either location. On the other hand, she has clearly rated different types of music during mornings compared to evenings, regardless of the weekday. Using this information, the time dimension can be used to generate recommendations based on time.

As we can see, it is not always clear whether a contextual dimension can be used as basis for recommendations until analysing the ratings in the underlying base dimensions, in this case users and tunes. The useful dimensions can be determined by using feature selection methods, as noted in Section 2.2.

Suppose that our third dimension is usable for recommendations. We can then reduce the three dimensions into the two base dimensions by only considering the user-item ratings where the value of the third dimension is fixed. For example, we want to recommend tunes to Sarah to listen to on Saturday. In that case, we only consider the user-item ratings from the three-dimensional rating space where the time dimension has the value "weekend". Using those ratings with a traditional two-dimensional recommendation algorithm, the result will be predicted ratings for tunes suitable for Sarah on Saturday.

Another property inherited from the OLAP model is the support for aggregation hierarchies for different dimensions. As noted in section 2.1, context data is typically

hierarchical, depending on the level of aggregation of context data. For example, a location context could consist of aggregation levels *co-ordinate*, *street*, *city* and *country*. Using this hierarchical structure of the context data together with OLAP techniques for selecting suitable aggregation levels, we can choose the granularity of available context data that produces the most accurate recommendations. This results in an enhanced multi-dimensional recommendation model consisting of each dimension's attribute profile, the multi-dimensional cube of ratings and an aggregation hierarchy associated with each dimension.

Although the multi-dimensional model successfully generates context-dependent recommendations by reducing the total amount of ratings to only those given in the relevant context, the fact that it potentially discards most of the given ratings will require a larger collection of ratings than in a traditional recommendation system. Furthermore, the ratings will have to be evenly distributed over all context dimensions in order to generate context-dependent recommendations. Evaluations of the multi-dimensional approach [ASST05] show that it can give more accurate predictions in cases where context matters. However, since in any situation some context dimensions can be more significant than others, the selection of relevant dimensions to include in the dimension reduction is crucial for this method's accuracy.

3.6.2 Context Weighting

Instead of reducing the ratings to only the set given in the relevant context, the context information can be used as a *hint* for which ratings the recommendation algorithm should give more attention to. Chen [Che05] explores a possibility to use context similarity measures as weights for ratings in the two-dimensional recommendation model. In this model, the phases of the traditional recommendation process, from building a user profile to calculating similarity and generating recommendations, are extended to incorporate context information.

Let a context snapshot be a tuple $F = (f_1, f_2, ..., f_m)$ where f_t represents some value, e.g., "Paris" for the context feature "Location", denoted F_t. In order to determine which contexts are similar to each other we need some kind of quantifiable *similarity measure* for each context feature. This can be achieved by creating a customised comparison method for each context feature, which takes two context values, possibly from different aggregation levels, and returns a numerical similarity value. Denote this similarity measure $sim_t(f, g)$, where F_t is the feature, and f and g are the comparable values ($f, g \in F_t$).

If calculating the similarity is not feasible, a measure for *context relevance* can be obtained by taking the fundamental idea from collaborative filtering, namely that users whose ratings correlate over different items are related, and applying it to the concept of context similarity. Specifically, we can use the Pearson similarity measure, see Equation 8 in Section 3.4, to calculate the correlation between two contexts with respect to their user-item ratings by considering the item ratings from all users in the specific contexts.

Having obtained a measure for context similarity or relevance, we can weight the user-item ratings with either one or both of these values. This weight can then be used in the recommendation prediction process for determining how relevant a rating in one context is to a rating in the current context.

Now, let

$$R_{uif} = k \sum_{t=1}^{m} \sum_{g \in F_t} r_{uig} \cdot sim_t(f, g) \tag{10}$$

be the weighted ratings by the active user u for item i in the current context f, where context similarity have been used for weights. Here k is a normalising factor, such that the absolute values of the weights sum to unity, and r_{uig} denotes the rating by the user u for the item i in the context g. Note that the sums are nested, the outer iterating over context features, like *time* or *location*, and the inner over values in each feature, for example, *Paris* and *Helsinki*, for the contexts in which the user has rated item i.

We can now use the ratings R_{uif} in place of the weighted user-item ratings $sim(u, v)$ in the traditional collaborative filtering algorithm presented in Equation 7 to generate context-dependent recommendations. This is the method of choice for creating context-aware recommendations in the roadmap for AndroMedia.

4 Survey of Context-aware Mobile Music Recommenders

In the previous sections, we got familiar with the concepts of context-awareness in mobile systems and derived a context-sensitive recommendation method. In this section, we will explore some of the context-aware mobile music recommender systems found in the literature. Note that although there are many commercial music recommendation systems available, we only consider music recommenders for a mo-

bile setting that make use of some form of context information. The systems are divided into two groups: those that only sense nearby devices to communicate with, and systems that incorporate more sophisticated context data. A summary with technical details of these systems as well as AndroMedia is presented in Table 4 at the end of Section 5.

4.1 Proximity Recommenders

This class of mobile recommendation systems are based on sensing other nearby devices running the same system and communicating with them. Recommendations are based on exchanging tunes or playlists between devices and can comprise two or more devices simultaneously.

4.1.1 BluetunA

BluetunA is a collaborative media application for mobile phones [BJBW07]. As the successor of tunA [BMA06], which is based on Wi-Fi connectivity and operates on a PDA, BluetunA shares most of the features with its predecessor. However, unlike tunA, BluetunA runs on mobile phones for a couple of reasons. First, mobile phones are more common and have become powerful enough to run more advanced software. Second, since many mobile phones now support playback of MP3 files, BluetunA developers believe that users are more inclined to listen to music on their mobile phones than carrying around a separate media device. This change, together with the fact that Bluetooth is a more common feature than Wi-Fi in mainstream mobile phones, has led to a change in means of communication between nearby devices from Wi-Fi to Bluetooth. Additionally, TunA was designed to share music between users, whereas BluetunA only send metadata from one device to another.

BluetunA allows users to share a list of their favourite tunes with other nearby users. BluetunA either generates the list of favourites using the tunes that are currently on the device, or connects to the Last.fm[5] service on the Internet and uses the user's personal profile to generate the list. The former method yields a list of favourites that describes the user's musical preferences at a shorter time span since the music on the device normally changes over time as the user seeks musical variety. The list derived from the user's Last.fm profile describes more closely the user's general music taste, as the information is gathered over a much longer period of time. By

[5]Last.fm tracks user's listening habits and calculates recommendations based on user similarity.

publishing either one of these lists, the user can choose whether to inform other users about the current listening habit or a general music taste. The connection to the Last.fm service also allows the users to receive recommendations. These recommendations are not, however, based on physically nearby users, but on the user's listening habits compared to other Last.fm users.

By utilising the Bluetooth connectivity of mobile phones, BluetunA lets its users interact with each other by exchanging playlist information or instant messages when in close proximity. BluetunA includes a search function that can be used to search for other BluetunA users nearby. The user can also let BluetunA search for nearby users with similar taste in music using a set of keywords describing the type of music sought for. The search can either be performed automatically on regular intervals or manually at any particular time. Upon displaying the found nearby users, BluetunA distinguishes between friends, total strangers and familiar strangers. Familiar strangers are people whom the user does not know but whose paths the user has crossed some times before. The algorithm used to find matching users calculates a weighted sum of the normalised matching between the locally input search terms and the metadata tags of the other user's MP3 files as well as between both users' favourite artists.

The developers recognise the difficulty to develop mobile applications for different mobile phones, and as such, the prototype has only been tested on two different mobile phone models.

The motivation behind the development of BluetunA as a music taste sharing application is that people are genuinely interested in strangers' musical preferences. The developers recognises an increase in Internet sites that support sharing information about personal interests, such as Del.icio.us, MySpace, Last.fm and Flickr, and believes that this interest in sharing personal information and viewing other peoples' preferences also transfers to the mobile short-range setting. However, the fundamental difference between these systems is the number of users a user is able to interact with. On an Internet site with potentially millions of users, the diversity of interests and the possibility to find similar users is much greater than when viewing the musical preferences of a handful of users that happen to be in proximity. Hence, viewing nearby peoples' tastes in music seems to merely satisfy curiosity. BluetunA's utilisation of the Last.fm recommendation service seems a more lucrative approach towards receiving new appealing music.

4.1.2 SoundPryer

The developers of SoundPryer [Öst04] aim at enhancing the music listening experience while driving a car. The idea of SoundPryer is to add value to the numerous traffic encounters by letting the user listen in on what music the encountering drivers are listening to.

Essentially, SoundPryer is a shared car stereo that operates in one of three modes: local play, remote play or automatic. In local play, the device has the same functionality as a normal music player, except that it also broadcasts everything that is being played. Devices in remote play mode can tune in to the stream being broadcasted by a device in local play mode. In automatic mode the device plays locally until a broadcasting device is discovered and then switches to remote play.

SoundPryer runs on Wi-Fi enabled PDAs and uses mobile ad-hoc networking technology to form a self-organized decentralised network of devices. Nodes can freely form small networks and enter or exit the networks as devices enter their proximity. As traffic encounters are generally short-lived, they set high requirements on the network protocol used for communication between devices. The time to discover new nodes in the network is generally very limited and SoundPryer must further ensure that all peers in the network are aware of each other.

Since the device is used in situations where the user is busy, the interface must be easy to use in order not to endanger the driver. The PDA is mounted to the dashboard to give the driver easy access to the interface. The actual interface is simplistic with large buttons allowing the driver to use fingers instead of a stylus to operate the device. The interface design also allows the user to get all information from the screen with just a quick glance. The interface has two different looks depending on the mode of operation. The local play interface consists of large buttons to control playback and an area for displaying information about the currently playing tune. In remote play mode the interface only consists of a simplified visual representation of the vehicle broadcasting the stream including vehicle type and colour. This helps the user identify the broadcasting vehicle in a situation where the user has several other vehicles around him. No information about what music is being played is conveyed.

The SoundPryer concepts seems somewhat similar to using Apple's iPod in cooperation with the third party accessory iTrip[6]. With iTrip, an iPod can broadcast

[6]http://www.griffintechnology.com/products/itrip/

music over the FM band instead of transmitting it to the user's headphones. The broadcasted music can be tuned in using any FM radio, for example, the car radio. If other drivers comes in range of the transmitted FM radio signal, they can also tune in on the broadcasting frequency. However, with iTrip, there is no way of knowing if there is a broadcasting device nearby, unless scanning the whole FM frequency range, in which case the iTrip broadcast would appear as any normal radio station. SoundPryer, on the other hand, tunes in to broadcasted music automatically. The SoundPryer concept appears to be most useful in city driving where the distance between cars are small and two cars can stay close to each other for a fair amount of time.

Even though SoundPryer is designed for use while driving a car, the interface design, with large buttons that can be interacted with using fingers, is suitable for any mobile setting where the user can only pay limited attention to the device, and the device lacks tactile feedback. In AndroMedia, we have taken the same approach, making user interface elements, with which the user interact often, large enough to use without a PDA stylus.

4.1.3 Bubbles

Bubbles is a wireless peer-to-peer music sharing application for mobile devices [BBFB+03]. The development of Bubbles has been guided by an ethnographic study suggesting that people's goals are continuously redefined and re-evaluated to accommodate unforeseen events and surprising actions. In other words, people are willing to change their daily routines based on events in their surroundings, adapting to the current context and seize profitable opportunities. This type of opportunism and spontaneity are key elements in the design of Bubbles

Each Bubbles device is thought of as being surrounded by a sphere whose radius depends on the range of the wireless network card of the device. As devices closely pass each other, ad hoc wireless networks are continuously formed and dissolved in an unpredictable manner. When two or more peers form an ad hoc network in the Bubbles concept, their corresponding bubbles are seen as overlapping, and the peers can communicate with each other. This spontaneity, which is created as devices suddenly are in communication range, is one of the key concepts in the design of the Bubbles prototype.

The other key concept, opportunism, is implemented by letting the user act upon

the information conveyed by the other peers. Since Bubbles is a media sharing application, the information transferred between devices is mainly related to music and pictures, but some information about the user, such as an avatar, can be made available for other users. When two bubbles overlap, the users can act upon the shared media information according to their own interest or current situation.

A prototype of Bubbles has been implemented on a Wi-Fi enabled PDA. The prototype illustrates the concept of information bubbles as nearby circles containing an icon representing the user of the device. The prototype also includes a context-aware toolbar and a ticker displaying sample items from the union of all remote playlists in the user's proximity. This ticker serves as a springboard for opportunistic behaviour as the user is presented a random sample independent of remote user information. By clicking on an item in the ticker, the user can choose to either stream the tune directly from the remote device or download it.

By clicking on one of the bubbles denoting another nearby user, a view containing additional information about the user, such as an avatar with accompanying greeting, is revealed. From this screen the user can initiate a chat with the other user and enable a third view displaying the whole list of available tunes on the remote device. Through the playlist view the user can stream or download single MP3 files or download the entire playlist to the local device.

The developers identify some potential problems with the prototype. There is a risk of exposing the users to an information overload, especially when they move about in crowds. When a large number of devices are nearby, they all form a common ad hoc network which potentially has huge number of media items. These kind of problems can be solved by creating user profiles, to which a filter can be applied to sort out less interesting items. However, filtering works against the idea of spontaneous and opportunistic media sharing. Instead of performing user profile based filtering, the developers of the Bubbles concept suggest an extension to the concept that would, according to appropriate heuristics, randomly and continuously select media items available on remote devices. This approach would present a stream of media items only visible for a short time and would thus also support the idea of user spontaneity and opportunism by requiring the user to act quickly upon the currently visible items.

This requirement, however, reveals a contradiction in the Bubbles concept. Spontaneous and opportunistic behaviour requires a certain degree of attention, something that the mobile user not necessarily is able to give the system as she is occupied with

other things. This limitation has currently no solution in the developed prototype application. Another problem with the Bubbles prototype are the legal implications of sharing music between devices. If users have copyrighted material on their devices, Bubbles would support piracy and thus would not be viable as a commercial application.

4.1.4 Push!Music

Push!Music [JRH06] builds on the idea that music could independently move from device to device instead of requiring users to actively send or download music. The developers introduce the concepts of media agents and media ecologies [HJH05]. For each media item on the device there exists a corresponding metadata file. The item and the metadata file associated with it form a *media agent*. Since playlists often contain media items that correlate in terms of, e.g., genre and are appreciated by the user as a collection, the media agents can learn the type of surrounding they are appreciated in. The developers of Push!Music calls this media environment a *media ecology*.

The media agents collaboratively form a model of the users musical preferences. By using the model as a basis for collaborative filtering, the agents can intelligently insert themselves into local playlists if they discover a playlist as a suitable media ecology. Network connectivity also enables the media agents to copy or move themselves to nearby devices. This makes the media items into intelligent agents that observe and act upon their environment, moving or copying themselves between playlists and devices. They can even delete themselves from the device if disk space is needed for more suitable agents. Although media agents move autonomously between devices, Push!Music still allows the user to deliberately *push*, i.e., send a media item to another user nearby.

The media ecology concept is interesting since it is based on both the media agent's context and the user's proximity information. Another interesting aspect is that media files move themselves between devices. This could potentially create a distribution chaos, where files disappear from a user's device and new files appear, and if not restricted by some user preferences, this kind of unpredictable behaviour is not especially user friendly. The manual push feature also raises questions like whether a media agent deletes itself immediately if pushed into an unsuitable media ecology. Furthermore, can a device be conquered if a large enough media ecology is pushed onto it, wiping out the original inhabitants? These questions do not seem to have

any answers in the Push!Music prototype.

4.1.5 Summary of Proximity Recommenders

BluetunA, SoundPryer, Bubbles and Push!Music all share the same philosophy: to use proximity information to interact with other nearby users. All systems run on handheld devices like PDAs and mobile phones and use either Wi-Fi or Bluetooth to communicate with devices in close proximity. Although most of the applications also provide services like instant messaging, image transfers and public user profiles, their main focus is to provide means for the users to communicate their musical preferences to other users nearby. This is done by either sending a list of tunes, streaming music or sending the actual media file to another device. If the media file is transferred to a nearby device, the application should include some means for copyright management. The developers of Push!Music suggest a *superdistribution* approach [MK90], where media items can be freely distributed but only used for a limited period of time unless paid for.

In AndroMedia, nearby devices are not communicated with, but only used as a dimension in the user's context. The information about nearby Bluetooth devices can be used to infer the user's company, which can have significance when recommending music based on the user's context. The information about other user's musical preferences is instead stored centrally and is indirectly available to all users through the recommendation mechanism.

4.2 Advanced Context Recommenders

Recently, new types of systems that take into consideration other kinds of context than proximity have emerged. When enhancing the above type of systems with sensor information, we get applications that are able to act upon changes in the physical environment of the user.

4.2.1 IM4Sports

IM4Sports [WPVS05] is a portable adaptive music player for use during sport exercises, mainly running. It attempts to motivate the runners as they often prefer to stay in pace with music while exercising. IM4Sports aids users in their exercise goals by either letting the music tempo follow the user's stride tempo or by inspiring

the user to keep in pace with the music. IM4Sports provides a playlist suitable for the exercise at hand, depending on the musical preferences of the user and other constraints.

The whole system comprises a PC, a portable music player, a heart rate waistband sensor and a pedometer. The PC contains a music library, a subset of which is used on the portable player during exercise. IM4Sports requires an initial set up process during which the user has to tag the whole music library with metadata in order for the system to function optimally. The metadata that is needed includes id, artist, genre, duration, file size and beats per minute (BMP). Also the user has to set up a personal profile which consists of the user's physical attributes such as name, gender, age, weight, resting heart rate, and peak heart rate.

The usage cycle is a three step process including preparation, exercise and feedback. The PC is used before and after the actual exercise, and the wirelessly connected pedometer and heart rate sensors are used with the portable player during the exercise. During the preparation, the user selects a training programme and downloads a suitable set of music to the portable music player. Since different training programmes have different characteristics in terms of tempo, duration and variation, also the music used during the exercise has to be customised. A training programme consists of different sub-programmes, such as warm-up, stretch, run, recovery and cool-down. The user can define the sub-programmes and their properties, for example, duration and target heart rate. When the user selects a training programme, IM4Sports uses the properties of the programme to select a suitable set of songs for the portable player.

IM4Sports generates a suitable playlist for the chosen exercise using a set of constraints. Besides the user's musical preferences, the constraints consist of *range*, *exclude*, *counting*, *summation*, and *all-different*. The range constraint ensures that the tempo of the tunes chosen for a certain exercise is in the correct range according to the exercise. If the user dislikes a tune that is being played during the exercise, she can apply an exclude constraint which will prevent the tune from being played again during the particular exercise. There is also a possibility to express the opposite opinion about a tune by applying a counting constraint which will restrict and force the number of different genres and/or artists in the generated playlist. The summation constraint guarantees that the duration of the generated playlist is long enough to cover the whole exercise. It also ensures that the total file size of the music items in the playlist does not exceed the amount of available space on

the portable device. Lastly, the all-different constraint ensures that all tunes in the playlist are different and that one tune will not be played twice during an exercise.

The process of generating a suitable playlist is merely a task of selecting tunes that satisfy all constraints. To solve this combinatorial problem, IM4Sports uses approximative local search, which translates the constraints into penalty functions. A penalty is set when a constraint is not met, and the total penalty for a media item is the weighted combination of all penalties. Weights indicate the severity of violating the constraint. To find a suitable playlist that minimises the total penalty, the local search considers the whole set and iteratively adds and/or removes an item from the set until an acceptable set has been found.

During the actual exercise, the user's heart rate and stride frequency are conveyed to the player using a waist strap and a pedometer. The system uses a five point average value for both heart rate and stride frequency and either guides or supports the user in the exercise in accordance with the training programme.

Besides this linear play, IM4Sports also comes with some advanced playback modes. In pace-fixing mode, the player keeps a constant tempo which encourages the user to stay in pace with the music. Since the tempo generally differs between tunes, a *time stretching* method can be used to adjust the tempo of a tune by shortening and lengthening the tune without a change in pitch. This can be done to a degree of -15% to +25% without remarkable distortion. Music items whose tempo most closely matches the required pace are preferred when the player chooses the next tune.

In pace-matching mode, the user dictates the tempo while the player adapts the music to follow it. When the user speeds up, the tempo follows and when the user slows down, so does the music. This mode also uses the time stretching method to adapt the tempo of the music to the stride frequency of the user. If the music item cannot be stretched to the required tempo within the acceptable margins, the system chooses a new tune whose tempo more closely matches the needed tempo. The transition between the tunes are made seamless by adjusting the tempo of the original tune to its limit, then switching tune and playing the new tune starting at the same tempo. The new tune will then speed up or slow down to the required tempo.

In pace-influencing mode the tempo of the music changes to keep the users heart rate within a certain range. This is done in a four-step procedure where first the tempo of the music is matched to the user's stride frequency to ensure that the user

follows the tempo. By comparing the user's heart rate to the current tempo, the system can calculate the required tempo and stride frequency for the target heart rate, as defined by the training programme, and successively adjusts the tempo to match these values. This process continues throughout the training programme.

When the exercise is over, the user can download the collected data about the exercise session from the portable device to a PC for visual inspection and system learning. The user can inspect the logged heart rate to evaluate the success of the training session. The system uses the stride frequency data from several exercise sessions to determine the selection of optimal tunes for the exercise.

IM4Sports is based on the idea that music can have a positive effect on training results, as presented in the Introduction. By adjusting the tempo of the music, IM4Sports can either spur the user to keep a certain pace, or follow the pace determined by the user. As opposed to AndroMedia, the recommendation mechanism of IM4Sports is based on predefined activity. The user needs to select a training program prior to the actual exercise and download a suitable set of music to the portable player. The context data used in the music recommendation process is thus explicitly input by the user. The context data implicitly collected during the exercise is used partly for manipulating the played music according to the chosen playing mode during exercise and partly for system learning after the exercise. In contrast, AndroMedia uses context data in real time as part of the actual recommendation process.

One of the crucial pieces of metadata in IM4Sports is the beats per minute value, since it is needed to make the transition between tunes seamless. If this piece of information is not included in the file, which is usually the case, the user needs to input the value manually for each tune. Unless a third party application can be used to automate the tagging, this can be quite a laborious process that might scare off the user. In AndroMedia, we have minimised the effort required by the user to start using the system. Except from the often initially occurring metadata *artist* and *title*, no tagging is needed. The user also does not need to use a desktop computer for anything else than transferring the tunes to the device, a task that every portable player requires.

4.2.2 LifeTrak

Lifetrak is a context sensitive music engine [RM06] that uses sensor data combined with user preferences to produce a customised playlist for a certain situation. The hypothesis behind the development of Lifetrak is that the user's environment affects what kind of music she wants to listen to.

The context data is gathered on even and configurable intervals. It is represented by a set of tags, each denoting a pre-defined value of a certain dimension of the context which initially are assigned a weight of 0. The raw sensor data for each dimension in the current context is mapped to one of these tags, which gets the weight 1. The resulting context vector thus consists of all possible tags, each having a weight of 1 or 0 depending on the applicability of the tag to the current context.

For example, suppose that the system is able to sense two context dimensions, location and time. Location can have the values "home" or "work" and time can have the values "morning", "afternoon" or "evening". At fist, all five values have the weight 0. If the current context is sensed to be "home" and "morning", these values would be given a weight of 1.

In Lifetrak, the sensed environment consists of the following dimensions: *space*, *time*, *kinetic*, *meteorological* and *entropic*. Space, i.e., location information is obtained using an external GPS sensor. The co-ordinates are mapped to US zip codes using a geocode data set from the US Census Bureau[7]. According to the developers of Lifetrak, an approximate mapping from co-ordinates to zip codes is sufficient, even though the resulting regions can be quite large and inaccurate. Since GPS information is restricted to outdoor environments, the location context can also have a value of "indoor" or "outdoor" based on GPS data availability. In the case the GPS data cannot be obtained, the last known GPS co-ordinates are used for zip code mapping. The local time is provided by the device itself and is mapped into four periods: morning, afternoon, evening, and night. The time dimension also includes the day of the week. The kinetic dimension consists of the velocity of the user and is calculated by the GPS sensor, thus also limited to outdoor usage. The velocity is mapped to four pre-defined values: static, walking, running, and driving. The entropic dimension, i.e. the state of the environment, is calculated in one of two ways. If the user is driving a car, the entropy is collected through RSS feeds describing the traffic conditions in the area. If the user is not driving, a five seconds

[7] http://www.census.gov/geo/www/tiger/

audio sample is recorded using the device's microphone. The overall decibel level of the sample is used as a measurement of the busyness around the user. Lastly, the weather information is gathered from a Yahoo.com weather RSS feed. The temperature data is mapped into one of five values: frigid, cold, temperate, warm, or hot.

Lifetrak does not attempt to analyse tunes in order to match them to a certain context. Instead, the system requires an initial set up prior to usage which includes tagging the whole music library. The tags include id, file name, artist name, tune name, album name, ranking, and context. The user can select any combination of available context tags to denote the context in which the tune is preferred. The complete set of available context tags form a vector with values of either 0 or 1, depending on the tags being selected or not. The values can later be modified through usage to become any real number between 0 and 1.

During usage, Lifetrak generates a playlist "on the fly" by matching the current context vector to the context vector of each tune. The matching is done by calculating the dot product of these two vectors, resulting in a value between 0 and the total number of available tags, i.e., the length of the vector. This integer is saved as the ranking of a tune in the current context. All media items are ordered by ranking and played from highest to lowest ranking value.

The user interface of Lifetrak allows the user to give feedback to the matching algorithm. The user can modify the values of both the current context vector and the tunes context vector by different means. The suitability of the currently playing tune in the current context can be modified using Love it or Hate it buttons. These buttons increase or decrease, respectively, the value of the tune's context tags that describe the current context. The influence of the current context can be modified using a *context equalizer* [RM06]. The equaliser consists of sliders for each dimension of the context, each increasing or decreasing the effect of that dimension in the ranking algorithm. The modifications of the context vectors can be seen as additional factors in the dot product, which after user adjustments consists of the original context vectors and two adjustment vectors acting as weights for the context values.

Lifetrak is similar to AndroMedia in many senses. First, Lifetrak is based on the same hypothesis as AndroMedia, i.e., that users' context affects what they want to listen to. Second, Lifetrak's context-awareness is purely based on sensor data, which is similar to the approach in AndroMedia, where sensor data comprises the majority of the context input. Last, both Lifetrak and AndroMedia use binary ex-

plicit user feedback. However, Lifetrak bases its music recommendations on context tags associated with each media file. For optimal operation, the tags need to be manually entered for each media file prior to usage. A side effect of allowing the user to explicitly mark a tune as suitable or not for a specific context is that the system can let the user bypass the potentially laborious tagging process by instead using the explicit user feedback to gradually populate each tune's initially empty context tags.

Lifetrak generates a playlist according to the tunes' relevancy ratings in the current context, and plays the tunes in ranking order. Lifetrak does not, by design, allow the user to view the generated playlist or skip to a certain song. Instead the user is forced to follow the pre-determined playing order. In contrast, AndroMedia is designed to function as any conventional portable media player, allowing the users to compose their playlists according to their own preferences and skip songs in whatever way they see fit. The context-aware recommendations are merely an optional feature that the user can employ to get suggestions on suitable songs in the current situation.

5 AndroMedia

We now present AndroMedia, a mobile context-aware music recommender prototype. The system consists of a client and a server, both of which will be presented in subsequent sections.

5.1 The AndroMedia Client

First, we present the AndroMedia client, the end user's interface to the AndroMedia system.

5.1.1 Overview

The AndroMedia client is a handheld media player, which aims at functioning as any other modern mobile media player in terms of usability and basic functionality. However, the context data gathering tool BeTelGeuse [NKL+07] has been integrated into AndroMedia to provide it with context data. By combining the context data with users' interaction data, AndroMedia is designed to recommend suitable tunes for the user in the current context using context-filtering techniques, as described

	BluetunA	SoundPryer	Bubbles	Push!Music	IM4Sports	LifeTrak	AndroMedia
Device	Mobile phone	iPAQ PDA	PDA	PDA	?	Nokia 770	iPAQ PDA
Operating system	?	PocketPC 2002	PocketPC 2002	PocketPC 2002	?	Linux	Windows Mobile 2003 SE
Programming language	Java	C / C++	C++	?	?	C, Python	C#, Java
Libraries	JSR-75, JSR-82	libmad MPEG, game GUI	Fmod, GapiDraw, OpenTrek	Fmod, GapiDraw	?	GTK+, Hildon UI, GStreamer, libid3tag	Windows Media Player Mobile
Device communication	Bluetooth	WLAN	WLAN	WLAN	None	?	None
Availability[a]	WWW	PROTO	PROTO	PROTO	PROTO	PROTO	PROTO
Context information	Proximity	Proximity	Proximity	Proximity	Heart rate, stride frequency	Location, time, velocity, meteorological data, traffic conditions, ambient noise	Over 12 properties, see Section 5.1.2

Table 4: Comparison and technical details of explored mobile music recommenders. Values that are not evident from the literature are denoted with a '?' throughout the table.

[a]WWW = available as download on the Internet, PROTO = research prototype, not available as a product.

in Section 3.6.

The target platform of AndroMedia is an HP iPAQ HX 4700 personal digital assistant running Windows Mobile 2003 Second Edition. The iPAQ comes with a 4" VGA touch-sensitive display, built-in WLAN 802.11b and Bluetooth connectivity hardware and expansion slots for Secure Digital and Compact Flash memory cards. Additionally, Windows Mobile 2003 SE comes bundled with Windows Media Player 10 Mobile for PocketPC, which includes an ActiveX controller that can be used in other applications.

The hardware and software features of the iPAQ were important factors in the choice of target device. The development of the client started in January 2006, and since affordable mobile phones at that time did not provide the required environment in terms of programming interfaces such as Python or Java, we believed that a PDA would serve better as development platform. Although the iPAQ lacked GPRS hardware, it still provided suitable means of wireless communication, such as Wi-Fi and Bluetooth. According to specifications, the bundled media player was also able to host third party plug-ins and could be used as a component in third party .NET applications by generating a wrapper around it, dealing with interoperability issues between the older COM objects and the new .NET platform. These features made the iPAQ a suitable device for the AndroMedia client, since we were looking for a way to extend an existing media player application.

During the development of the client, however, we noticed that Windows Media Player Mobile 10 in fact did not provide part of the functionality we relied on, although the documentation claimed the opposite [Mic08]. Specifically, we were not able to use the component's built in media library, nor could we get the component to trigger events when interacting with its playback controls. There were also problems with the different user interface modes of the component, such as hiding the playback controls to make only the visualisation screen visible. In addition, the component was not able to read all of the embedded ID3 metadata of MP3 files, leaving us with only a small set of metadata to use for tune profiling. AndroMedia was originally supposed to support both audio and video, but due to the problems with the ActiveX component, we had to limit the media type to music.

It is not known to us whether these problems origin from flaws in the ActiveX controller itself or in the wrapper needed to host it in a third party application, but they forced us to reduce the use of the media player component to merely provide music playback and limited metadata extraction. Hence, the AndroMedia client

itself implements most of the application logic normally provided by the media player component. Specifically, components such as playback controls, playback logic, the playlist and media library and their functions have been written from scratch. Although this required more work, it ultimately allowed us to customise the application to our particular needs.

5.1.2 BeTelGeuse Integration

In AndroMedia, we use the context gatherer BeTelGeuse (BTG) [NKL+07] to obtain context information used by the application. BeTelGeuse is a tool for data gathering that uses bluetooth equipped sensors as data sources. We chose BeTelGeuse since it has been developed in the same research group as AndroMedia and since it is publicly available under the Gnu Lesser General Public License [LGP07]. This gives us the possibility to customise BeTelGeuse for AndroMedia's needs in terms of communication protocols and integration.

BTG consists of a client and a server, both written in Java. The client operates on mobile phones, PCs and other Java-compatible handheld devices with Bluetooth capabilities. It is responsible for managing the connection to available sensors and for collecting raw sensor data from the sensors. The data is either stored locally in a file for later submission or sent over GPRS or WLAN directly to the main server, which stores the data. BeTelGeuse does not provide any means for context reasoning; the interpretation of the gathered context data is left to the application designer.

In our laboratory setting, BTG can currently gather data from three sensors and the device itself. Using these sensors we can obtain the following data.

- **GPS** — latitude, longitude, altitude, time
- **Alive Heart Monitor** — heart rate, 3-axis acceleration
- **I-CubeX sensor system** — 1-dimensional distance, 3-axis acceleration, temperature, humidity, orientation, ambient light
- **Local device** — nearby Bluetooth devices; GSM information such as cell ID, area ID, network and country codes, network name, signal strength

BeTelGeuse communicates with AndroMedia over a local TCP socket, sending information messages about connected sensors. It also listens to incoming command

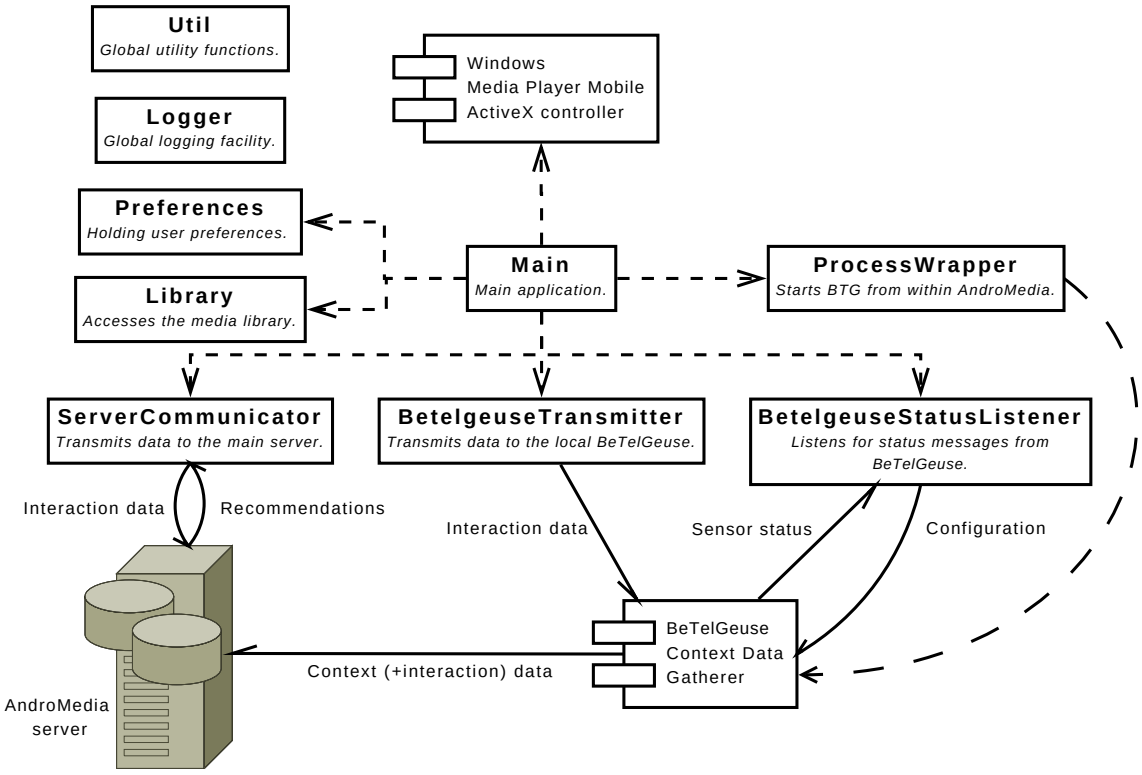


Figure 1: Component overview of the AndroMedia client.

messages, which allows AndroMedia to control the behaviour of BTG. Additionally, it provides means for AndroMedia to use BeTelGeuse as a proxy for sending user interaction data if needed.

5.1.3 Architecture

The AndroMedia client consists of a set of classes representing the various screens of the user interface, as well as a few communication and utility classes, see Figure 1. The application is built on top of the Windows Media Player Mobile ActiveX controller, which provides the media playback functionality. The `Main`, `Preferences`, `Library` and `Logger` classes all represent the corresponding screens in the user interface. The `Main` class incorporates the media player and provides most of the application functionality. The `Preference` class stores user preferences while the `Library` class provides means to browse and select music files available on the device. Both the `Preference` and `Library` classes use object serialisation to save their internal information to XML files on disk for persistence. The `Logger` class is used as a central logging facility by other components, and is responsible for storing the

logged messages to disk and show them to the user on demand.

The `ProcessWrapper` class is used to start the execution of BeTelGeuse, which handles all context data gathering. Since BeTelGeuse is a Java application, it has to be run inside a Java virtual machine, in our case the IBM J9 VM[8]. When BTG is executing, the `BetelgeuseStatusListener` class listens to a local TCP socket for incoming messages from BTG. The messages contain status information about the sensors connected to BTG to display to the user in the main interface. Each received message is answered with a response with which the user can control BTG's Bluetooth device discovery procedure.

The communication classes handle the communication between the client and the server. The client generates two sets of data: context data from BTG and user interaction data from AndroMedia, both of which are eventually stored on the server. The means of transferring the data to the server, however, depend on the availability of an Internet connection. AndroMedia operates in one of two modes, offline or online, depending on Internet connectivity. If no Internet connection is available, all data is stored on the device for later insertion into the database. In this case, the interaction data is sent from AndroMedia to BTG using the `BetelgeuseTransmitter` class. BTG stores both interaction and context information on the device in a file that can later be parsed into a database. When in online mode, the application is in direct connection with the main server and is then also able to receive recommendations. In this case, the interaction data is sent directly from AndroMedia to the server using the `ServerCommunicator` class, while BTG sends context data to the server by its own means. The two sets of data are connected in the database by a shared session id.

5.1.4 Operation

The interface is designed to be easy to use even without a stylus. This is achieved by making all elements with which the user interacts on a regular basis sufficiently large so that the users can push buttons, scroll lists and navigate menus with their fingers. Some user interface elements are, however, fixed in size by the operating system and cannot be changed. The main screen consists of a playlist window, a set of buttons for controlling playback, "Rocks" and "Sucks" buttons for explicit feedback, a set of icons representing context sensor status, and a main menu, see Figure 2(a). From the main menu, the user can navigate to the Preferences page,

[8]http://www-306.ibm.com/software/wireless/weme/

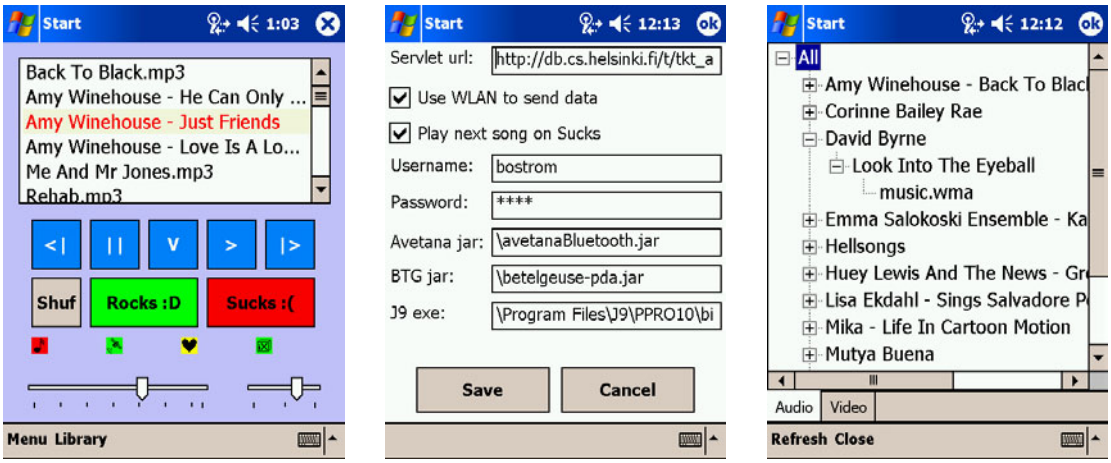
the Library and the Log, control the behaviour of the context sensor BeTelGeuse, request recommendations, and exit the system.

The playlist window contains a set of tunes that the user has selected for playback from the library. The playlist provides a context menu that pops up when the user taps-and-holds an item in the list. The context menu allows the user to remove the selected tune from the playlist or clearing the whole playlist, as well as to start playback of the selected tune. Playback can also be started by either double-tapping a tune in the playlist or by pressing the play button.

The "Rocks" and "Sucks" buttons provide users with means to give explicit feedback to the system regarding their musical preferences. A click on the "Rocks" button indicates that the user likes the tune in the current context, while the "Sucks" button provides corresponding negative feedback. Both "Rocks" and "Sucks" buttons can only be clicked once during the playback of any one tune, as they are both dimmed and disabled after a click on either one. The buttons are reactivated when the tune changes or is restarted.

The current status of the sensors connected to BeTelGeuse is conveyed to the user by using one status icon per connected sensor, under the "Rocks" and "Sucks" buttons, see Figure 2(a). The icon's visual representation depends on the type of sensor, such as GPS or Heart Rate Monitor. A change in sensor status is represented by changing the background colour of the icon to either green, yellow or red. A green icon represents a healthy sensor from which data is continuously received. If BTG has lost connection to a sensor, its corresponding icon turns yellow and if BTG considers the sensor to be dead, the icon turns red. A tap on an icon renders a textual explanation of the current state of the icon. The number and type of sensors connected and their status is received through the status messages sent continuously to AndroMedia by BTG. After each status message is received, the icons on the main screen are updated by adding new sensors and updating the status colour.

The Preferences page contains paths to the Bluetooth stack implementation library, the Java virtual machine executable and BeTelGeuse on the device, the URL and the user's username and password for the AndroMedia server, an option for storing the data locally or sending it over the Internet, and an option regarding the playback logic, see Figure 2(b). The settings regarding the operation of BeTelGeuse are only editable when the context gatherer is not running, as the same parameters must be kept unchanged throughout a context gathering session. When BeTelGeuse is running, these options are greyed out and disabled, and only enabled again when



(a) The main screen (b) The preference page (c) The library page

Figure 2: The AndroMedia client user interface

BTG is stopped.

The Library is an interface for searching and selecting media files available on the device, see Figure 2(c). The Library can scan a certain directory on the device for media files and will display all matches in a tree structured list. Each node represents either a directory (non-leaf node) or a file (leaf node). In order to add either a whole directory or a single file to the playlist in the main screen, the user selects "Add to playlist" in the context-menu appearing when the user taps-and-holds a node in the Library view. The user can add any number of items to the playlist, and even the whole library by adding the root node. AndroMedia does not automatically scan the device for new tunes, since the addition of new material is regarded a rather uncommon event. Thus, the library needs to be updated manually by tapping the Refresh button each time new media files are added to or removed from the device.

In order to provide context-aware recommendations to the user, the application needs to gather information about the user's interactions with the user interface. Each time the user performs a significant action, the specific action along with information about the application status, see Table 5, are sent asynchronously either directly to the server or via BTG, to use as basis for recommendations. An action is significant if it can be used to infer the user's preferences about the current tune or the user's current context, see Table 5 for examples. Among the application status information in Table 5, the current tune information is provided by the underlying Windows Media Player ActiveX controller and the session and playback information is provided by AndroMedia's classes.

When AndroMedia is connected to the Internet, the user can at any moment request recommendations from the AndroMedia server by selecting "Get recommendation" from the menu in the main screen. This sends a message to the server to calculate a suitable recommendation for the user given the current context. The resulting recommendation is displayed to the user in a pop-up window. If the recommended tune is available on the device, the user has the option of starting playback of that tune directly from the recommendation pop-up.

5.2 AndroMedia Server

The AndroMedia server is the back-end of the system. It stores all user activity data and context information in a database and uses this information to calculate recommendations for users. The back-end server also has a web interface through which users can inspect the data that has been collected about them by the AndroMedia client. In the following sections we will describe the functionality of the AndroMedia server and how it is implemented.

5.2.1 Architecture

The AndroMedia back-end server is written in Java Standard Edition and executed in the Apache Tomcat[9] servlet container on a Linux server. It uses a MySQL[10] database as persistent storage and XHTML[11] with CSS[12] for displaying the web-based user interface. It also features a command line tool that can be used for testing and debugging.

The back-end follows the Java Servlet specification, implementing two separate servlets, each of which extends the abstract class `javax.servlet.http.HttpServlet` of the Java Servlet package. The `MusicServlet` servlet is used for the web interface where users can inspect the information gathered about them by the system, while the `ScrobblerServlet` servlet is used for communicating with the AndroMedia client, accepting interaction messages and serving recommendations.

Both servlets provide their own specific operations, but use the same underlying stack of class layers to implement them, see Figure 3. Each HTTP request

[9] http://tomcat.apache.org
[10] http://www.mysql.com
[11] http://www.w3.org/TR/xhtml1/
[12] http://www.w3.org/TR/CSS21/

Significant actions	
Playback controls	Play
	Next (shuffle disabled)
	Next (shuffle enabled)
	Previous
	Pause
	Stop
	Volume change
Feedback buttons	Rocks
	Sucks
Playlist context menu	Play
	Jump (play by double-clicking a tune)
Automatic playback	Next (shuffle disabled)
	Next (shuffle enabled)
Application status information	
Session information	*Application*
	Session ID
	Username
	Challenge response
Current tune information	*Media type*
	Artist
	Song name
	Song length
Playback information	Playback position
	Current volume

Table 5: Interaction information sent by the client to the server. Italicised font indicates mandatory information.

starts off in the servlet classes, which forward the request to the `MusicRequest` or `ServletRequest` classes, respectively. These use a hierarchy of classes that each aggregates information on their own level. Immediately underneath the *request layer* lies the *task layer*, which consists of a set of classes representing a high level task, such as "add media item". These classes, in turn, use the *item layer* consisting of classes representing an abstract item, such as a "media item". Last, the item layer uses the *database layer* to communicate with the database. The database layer consists of one class for each table in the underlying database and is responsible for performing the lowest level database operations, such as reading and writing to the database.

Reversely, the data from the database is fetched by the database layer, which is transformed into high level items in the item layer. The task layer then uses the items in the item layer to perform its user level tasks and returns the request layer, which creates a response message suitable for returning to the client. The `ScrobblerServlet` class returns to the client a response string consisting of only a keyword and possibly a set of adjoining data. The `MusicServlet`, on the other hand, uses a set of Java classes representing XHML elements to build a web page to display to the user.

5.2.2 Database Architecture

The database layout has been designed to be as flexible as possible to allow many media types and different types of context data to be stored. Another reason for the generic layout is that several applications are using the same database for data storage. The database contains about 20 tables, which are logically clustered into three separate units, see Figure 4. The context data unit consists of tables specific to context data, such as sensors, sensor data, nearby bluetooth devices etc. These tables are governed by the context gatherer BeTelGeuse. The media unit, on the other hand, consists of tables containing data about media items, media types, metadata, user actions etc. These tables are not only used by AndroMedia, but also by Capricorn, an intelligent interface for mobile widgets [BFL+08]. The two main units are connected by a small set of shared tables storing session and user information.

When a user logs in, a new session for that user is created. As the user interacts with the application during a session, the media unit is filled with interaction data from AndroMedia, while the context unit is filled with context data from BeGelGeuse. The interaction data and context data are associated to each other and the user

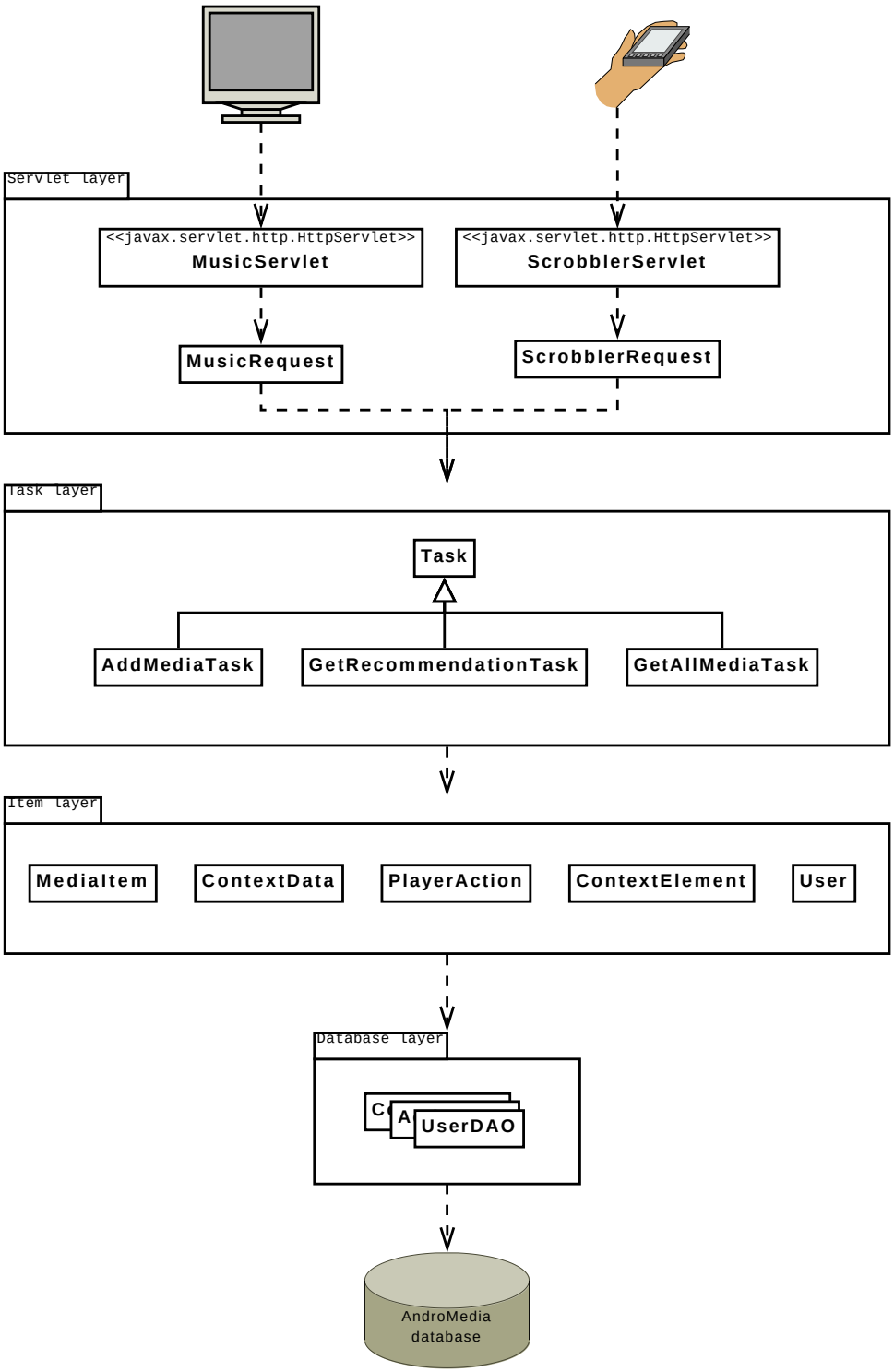


Figure 3: Component overview of the AndroMedia server.

through the session id, which is handed to both BeTelGeuse and AndroMedia when the session is created.

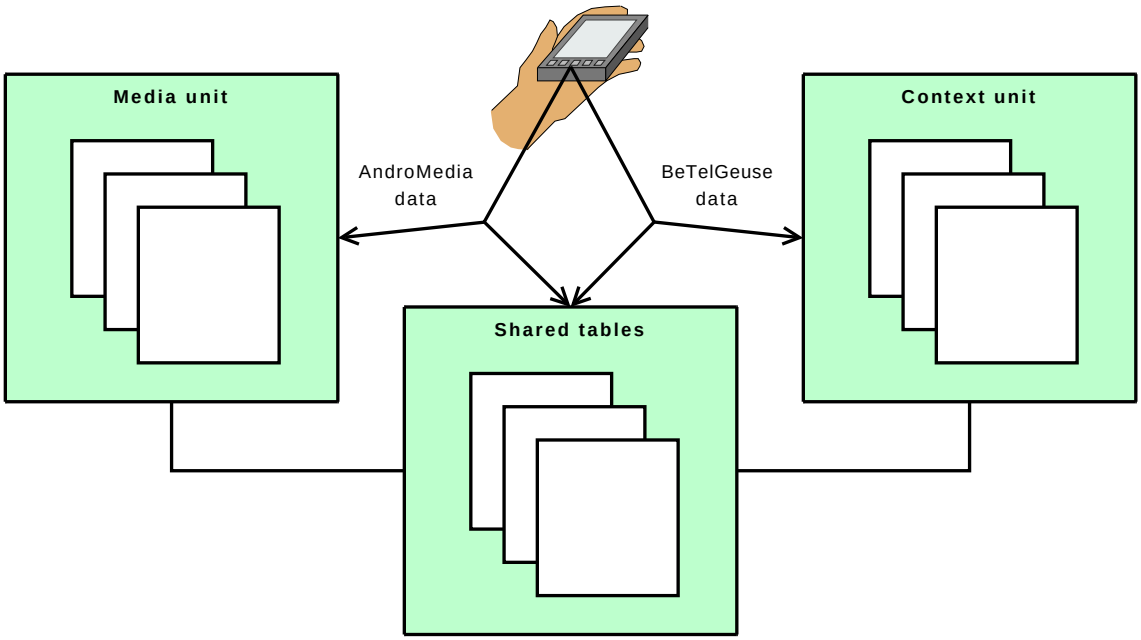
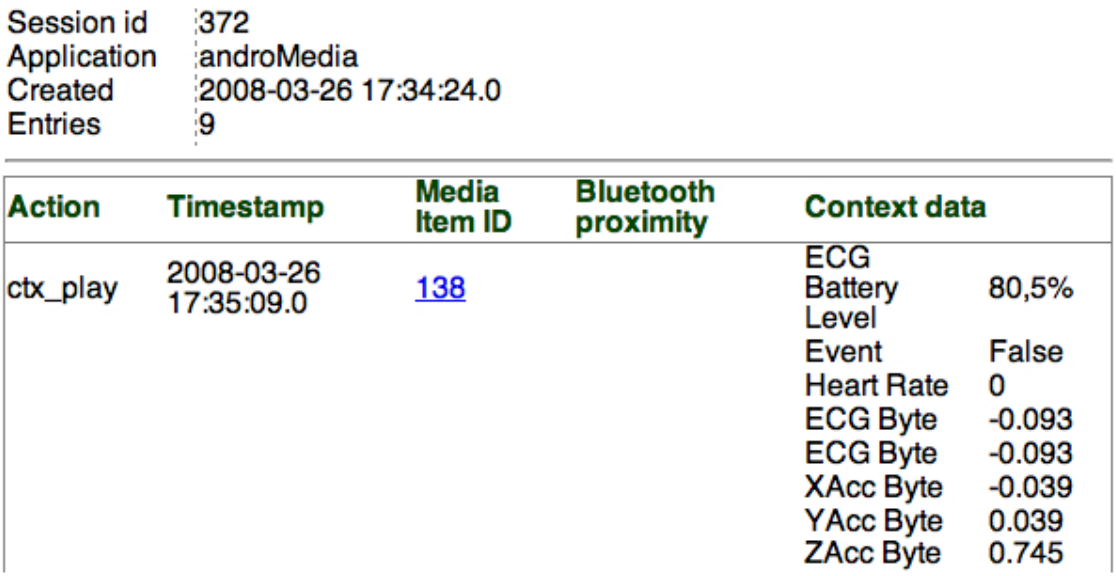


Figure 4: Overview of the AndroMedia database architecture.

Each interaction data unit consists of a user action and a media item acting as target for that action. An interaction data unit is represented in the database by a set of 9 tables, some of which describe the user action and others describe the media item upon which the action was executed. Each media item is seen as being of some media type, for example "music" or "video". Each media type is further associated with a set of metadata types that describe the item. In the case of a "music" item, the type metadata could include "artist", "album" and "title". Some of the media type specific metadata is used to identify a single media item, and is thus mandatory for that item. Continuing the above example, the "music" media type might have a mandatory metadata field "artist" and "title". A media item without the mandatory metadata information cannot be added to the database. When a media item is added to the database, the type of media is stored along with all available metadata for that item.

Similarly, an action is also assigned metadata, since some user actions are logged with parameters. A change in volume, for example, is logged as targeted at the currently playing tune with an action parameter describing the resulting volume level. The action and its metadata are logged in separate tables and linked together with the media item acting as target.



Session id	372
Application	androMedia
Created	2008-03-26 17:34:24.0
Entries	9

Action	Timestamp	Media Item ID	Bluetooth proximity	Context data	
ctx_play	2008-03-26 17:35:09.0	138		ECG Battery Level	80,5%
				Event	False
				Heart Rate	0
				ECG Byte	-0.093
				ECG Byte	-0.093
				XAcc Byte	-0.039
				YAcc Byte	0.039
				ZAcc Byte	0.745

Figure 5: Session view in the AndroMedia Admin web interface.

5.2.3 Functionality

As mentioned above, the AndroMedia back-end serves two purposes: to provide a web site for inspecting data gathered about the user, and providing storage and calculate recommendations for the AndroMedia client.

The AndroMedia Admin web interface servlet lets users inspect the data gathered about them by the application. The admin system consists of a front page, a sessions page, a media item page, a my music page and pages for registering to the system and for logging in. The front page displays the username of the user and some statistical information such as how many users and media items the system comprises. Using the "all sessions" page, the user can browse through all usage sessions. Each session is displayed with its id, the application that created the session, when it was created and how many entries the session contains. Selecting a session displays the entries for that session, see Figure 5. An entry consists of an action, the time the action was invoked, the id of the item upon which the action was invoked and the context data of that moment. The context data consists of the nearby bluetooth devices and other data depending on the attached sensors at that time. The user can further view information about the target item by clicking on its id in the entries table. This displays all available metadata, such as "artist", "title" and "album" in the case of an audio track.

The ScrobblerServlet provides an API for the client to save user interaction data and request recommendations. When a client connects for the first time during a session, it has to perform a handshake with the servlet. The client initiates the handshake by requesting a *challenge string* from the server. The server answers with challenge string which the client uses to calculate a *challenge response* using the MD5 Message-Digest algorithm. The response is generated by calculating the MD5 checksum of the concatenation of the challenge string and the MD5 checksum of the current user's password. The challenge response is then supplied with every action request as an identification token together with the user's username. If the client's challenge response is correct, the server authorises the action request.

When the user interacts with the client, it sends interaction data to the ScrobblerServlet as explained in Section 5.1.4. Some parameters among the information sent to the server are mandatory, as indicated by the italicised font in Table 5. The server runs a parameter check to ensure that there are no missing mandatory parameters before storing the information in the database.

The other service provided by the ScrobblerServlet is the recommendation service. Currently three different collaborative filtering algorithms are implemented: Pearson, naive Pearson and naive Bayesian, see Section 3.4.1. These do not take context information into consideration, but are used for testing purposes.

5.3 Implementation Status and Future Work

The functionality of the AndroMedia client and server presented in previous sections have been implemented successfully, and provides the framework needed for context-aware music recommendations. The client provides a fully functional mobile media player that can be used independently, and gathering context data is a matter of starting the context gatherer BeTelGeuse and connecting a set of sensors to it. Traditional recommendations can already be requested from the server, which also provides users with information about collected data.

Although the current implementation is mature and could be subjected to extensive user studies, the restricted amount of client devices at our disposal suggests that user studies should be postponed until a client application for a more pervasive device type, such as a Python client for mobile phones [ST07], has been developed.

The context-aware recommendation feature of the AndroMedia server will be implemented using the context weighting strategy presented in Section 3.6.2. Since

context data is already being gathered by the client, the remaining key phases in the development are:

- **Designing feature extraction methods.** The raw data collected need to be processed into homogenous feature vectors to be suitable for comparison. This process includes doing statistical calculations on numerical values, extracting multiple features from a single sensor and merging similar sensor data into more reliable features.
- **Develop similarity measures for advanced sensor types.** As discussed in Section 2.2, feature vectors need to be comparable in order to measure their similarities. This is achieved by defining a similarity operator on each feature type, and is trivial for simple numerical types. For complex types like composite or non-numerical data, for example, nearby Bluetooth MAC addresses and GPS co-ordinates, a custom similarity measure has to be defined.
- **Generating context-dependent recommendations.** Using the distance measure, the context weighting algorithm presented in Section 3.6.2 can be used to generate context-aware recommendations, which can be presented to the user.

6 Discussion and Conclusions

Since humans react to different types of music in certain ways, it seems feasible to consciously choose the type of music according to the situation. Many likely do so unconsciously by selecting their favourite music for certain situations, for example, up-beat music for training, classical music for studying and smooth jazz for driving. In an environment where an increasing number of everyday mobile devices have built-in music playback capabilities, music is likely to be continuously enjoyed over long periods of time, during which the user's environment is likely to change. A user may, for example, want to wake up to soft pop music, listen to jazz on the way to work, have rock music in the background while working, listen to jazz again on the way home and enjoy classical music in the evening. Even during shorter periods of time, like while shopping in the city, the environment changes from indoors with a low ambient noise to outdoors with a high level of noise and hectic tempo. During such changes in context, the playback management becomes a cognitive overhead that may distract or endanger the user. Mobile context-aware music players facilitate

the process of selecting and managing music for different occasions, decreasing the overhead and increasing the user's attention towards the matter at hand.

Context is a very wide concept that comprises every aspect of the situation of a user or a device, making it subject to very different interpretations. As noted in Section 4, many mobile context-aware applications interpret context as the collection of nearby devices and their properties. By being aware of devices in close proximity, the application can automatically initiate communication with other devices, sharing information and providing the user increased value. In the case of context-aware music recommenders, the focus is often on sharing music between similar users. This naturally has legal implications as copyrighted material is distributed to unknown peers, either with or without the user's knowledge and consent. As an alternative to copying the actual media file to the other device, like in the case of Push!Music, the music can be streamed to another device, letting the other user listen to a tune while the devices are in range. This approach does not illegally distribute reproductions of copyrighted material and can hardly be interpreted as illegal broadcasting since the range is limited and the time devices are connected is typically short. For devices with Internet connection capabilities, a legally feasible solution is to offer the user to buy the recommended tune from an on-line music store, if it is not already found on the device. As the online music industry has expanded from around 1 million tracks available in 30 legal stores in 2003, to over 6 million tracks available in over 500 online music stores in 2007 [IFP08], the probability to find any recommended tune purchasable online is steadily increasing. In AndroMedia, where an Internet connection is required to receive recommendations, this approach seems suitable, and is already partially implemented since the user has the option to play the recommended tune if it is found on the device.

Emerging artists that are trying to promote their music can also benefit from recommendation services. Even with a relatively small group of listeners, a recommendation system can deliver new music to listeners who find it appealing. With a total of nearly 3 million unsigned rock and R&B artists craving for attention on MySpace[13], one of the leading social network sites for music enthusiasts, the music industry recognises the need for recommendation and filtering services [IFP08].

More advanced context-aware applications that gather an extensive amount of situational data need to take into account possible privacy issues regarding collected data. Many context-aware applications have fallen due to privacy issues [May04],

[13]http://www.myspace.com

which indicates that this matter must be addressed when designing applications for usage outside the research lab. A plausible approach is to persistently save as little information as possible about the user, while still performing well in the context-aware features. In practice, excess sensor data could be deleted when no longer needed. The current policy of AndroMedia is to store as much information about to user as possible, and never delete it, to boost context-aware performance and to be able to recognise usage trends. In future work, privacy should be taken into account before doing extensive user studies.

During the development of the AndroMedia client, we have come to reconsider the choice of target device. Although it seemed appropriate in the beginning of the development, the HP iPAQ hx4700 PDA has a number of shortcomings as a context-aware mobile media player. First, the size and weight of the device (187 grams and 7.7 x 13.1 cm) exceeds the size and weight of modern MP3-players and mobile phones, making it somewhat clumsy to carry around and difficult to operate with one hand. Second, due to its touch-sensitive screen being the main interface for user input, the user can not operate the device without viewing the screen, since the screen lacks tactile feedback. Controlling playback and volume becomes cumbersome since the device has to be brought up and the screen cover has to be flipped over before the user can interact with it. A set of hardware buttons is available, and the use of these as input interface should be investigated. Third, the lack of GPRS connectivity makes the recommendation feature useable only near Wi-Fi hotspots.

In retrospect, a modern mobile phone would perhaps have been a more suitable target device for the AndroMedia client, as they address all of the shortcomings mentioned above. Especially, a client developed with mobile Python [ST07] that can be run on a mobile phone appears to be a feasible alternative to the current client. This technology was not available in 2006, when the development was started.

Besides the additional functionality discussed in the previous section, there seems to be room for improvement also on the server side. While the layered class architecture works well for small sets of data, initial results with growing data presages a performance issue. This can, however, be solved by deviating from the strict layered architecture and doing more complex database operations directly from higher levels of abstraction. This has been tried with some critical database operations, and the initial results look promising.

The AndroMedia Admin web interface could also be further developed to allow better management of user data. Especially, allowing users to delete their own collected

data could provide users with a higher sense of trust and would also have a positive effect on privacy issues. The recommendation feature could also be integrated into the Admin interface to allow users to explore what kind of recommendations they would receive in certain contexts. This would further facilitate downloading of recommended tunes from on-line music stores, as these are normally designed for use on desktop computers.

Although the AndroMedia prototype lacks an implementation of the context weighting algorithm needed for music suggestions based on the user's situation, the current implementation provides the framework needed for situation-sensitive recommendations and we believe it to be a firm step towards a context-aware mobile music recommender.

7 Acknowledgements

The author wishes to thank Jouni Sirén and Marja Hassinen for the development support on the AndroMedia server.

References

ASST05 Adomavicius, G., Sankaranarayanan, R., Sen, S. and Tuzhilin, A., Incorporating contextual information in recommender systems using a multidimensional approach. *ACM Transactions on Information Systems*, 23,1(2005), pages 103–145.

AT05 Adomavicius, G. and Tuzhilin, A., 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*, 17,6(2005), pages 734–749.

BBFB+03 Bach, E., Bygdås, S. S., Flydal-Blichfeldt, M., Mlonyeni, A., Myhre, O., Nyhus, S. I., Urnes, T., Åsmund Weltzien and Zanussi, A., Bubbles: Navigating multimedia content in mobile ad-hoc networks. *MUM '03: Proceedings of the 2nd International Conference on Mobile and Ubiquitous Multimedia*, 2003.

BC92 Belkin, N. J. and Croft, W. B., Information filtering and information

retrieval: two sides of the same coin? *Communications of the ACM*, 35,12(1992), pages 29–38.

BC08 Buttussi, F. and Chittaro, L., Mopet: A context-aware and user-adaptive wearable system for fitness training. *Artificial Intelligence in Medicine*, 42,2(2008), pages 153–163.

BFL+08 Boström, F., Floréen, P., Liu, T., Nurmi, P., Oikarinen, T.-K., Vetek, A. and Boda, P., Capricorn – an intelligent interface for mobile widgets. *IUI '08: Demonstration at the 12th International ACM Conference on Intelligent User Interfaces*. ACM Press, 2008.

BHC98 Basu, C., Hirsh, H. and Cohen, W., Recommendation as classification: using social and content-based information in recommendation. *AAAI '98/IAAI '98: Proceedings of the 15th national / 10th conference on Artificial intelligence / Innovative applications of artificial intelligence*, Menlo Park, CA, USA, 1998, American Association for Artificial Intelligence, pages 714–720.

BHK98 Breese, J. S., Heckerman, D. and Kadie, C., Empirical analysis of predictive algorithms for collaborative filtering. *UAI '98: Proceedings of the 14th Conference on Uncertainty in Artificial Intelligence*, 1998, pages 43–52.

BJBW07 Baumann, S., Jung, B., Bassoli, A. and Wisniowski, M., Bluetuna: let your neighbour know what music you like. *CHI '07: Extended abstracts on Human factors in computing systems*, New York, NY, USA, 2007, ACM Press, pages 1941–1946.

BMA06 Bassoli, A., Moore, J. and Agamanolis, S., tunA: Socialising music sharing on the move. In *Consuming Music Together: Social and Collaborative Aspects of Music Consumption Technologies*, O'Hara, K. and Brown, B., editors, Springer, 2006.

BMH95 Brownley, K. A., McMurray, R. G. and Hackney, A. C., Effects of music on physiological and affective responses to graded treadmill exercise in trained and untrained runners. *International Journal of Psychophysiology*, 19,3(1995), pages 193–201.

BMP04 Brown, S., Martinez, M. J. and Parsons, L. M., Passive music listening spontaneously engages limbic and paralimbic systems. *NeuroReport*, 15,13(2004), pages 2033–2037.

Bro01 Brodsky, W., The effects of music tempo on simulated driving performance and vehicular control. *Transportation Research Part F: Traffic Psychology and Behaviour*, 4,4(2001), pages 219–241.

BS97 Balabanović, M. and Shoham, Y., Fab: content-based, collaborative recommendation. *Communications of the ACM*, 40,3(1997), pages 66–72.

Bur02 Burke, R., Hybrid recommender systems: Survey and experiments. *User Modeling and User-Adapted Interaction*, 12,4(2002), pages 331–370.

Che05 Chen, A., Context-aware collaborative filtering system: Predicting the user's preference in the ubiquitous computing environment. In *Location and Context-Awareness*, Strang, T. and Linnhoff-Popien, C., editors, volume 3479, Springer, 2005, pages 244–253.

CHM97 Chickering, D. M., Heckerman, D. and Meek, C., A bayesian approach to learning bayesian networks with local structure. Technical Report, Redmond, WA, USA, 1997.

DEO+07 Dornbush, S., English, J., Oates, T., Segall, Z. and Joshi, A., Xpod: A human activity aware learning mobile music player. *IJCAI '07: Proceedings of the Workshop on Ambient Intelligence, 20th International Joint Conference on Artificial Intelligence*, 2007.

Dey01 Dey, A. K., Understanding and using context. *Personal Ubiquitous Computing*, 5,1(2001), pages 4–7.

Dow06 Downie, J. S., The music information retrieval evaluation exchange (mirex). *D-Lib Magazine*, 12,12(2006).

FW07 Floréen, P. and Wagner, M., editors. *Enabling technologies for mobile services: The MobiLife Book*, chapter 4, pages 99–151. Wiley, 2007.

GNOT92 Goldberg, D., Nichols, D., Oki, B. M. and Terry, D., Using collaborative filtering to weave an information tapestry. *Communications of the ACM*, 35,12(1992), pages 61–70.

GSCM07 Gauch, S., Speretta, M., Chandramouli, A. and Micarelli, A., User profiles for personalized information access. In *The Adaptive Web*, Brusilovsky, P., Kobsa, A. and Nejdl, W., editors, volume 4321, Springer, 2007, pages 54–89.

HHN04 Hargreaves, D. J., Hargreaves, J. J. and North, A. C., Uses of music in everyday life. *Music Perception*, 22,1(2004), pages 41–77.

HJH05 Håkansson, M., Jacobsson, M. and Holmquist, L. E., Designing a mobile music sharing system based on emergent properties. *AMT '05: Proceedings of the 2005 International Conference on Active Media Technology*, 2005.

HSRF95 Hill, W., Stead, L., Rosenstein, M. and Furnas, G., Recommending and evaluating choices in a virtual community of use. *CHI '95: Proceedings of the SIGCHI conference on Human factors in computing systems*, New York, NY, USA, 1995, ACM Press/Addison-Wesley Publishing Co., pages 194–201.

IFP08 The International Federation of the Phonographic Industry (IFPI) Digital music report 2008 - summary, `http://www.ifpi.org/content/library/DMR2008-summary.pdf`, January 2008. Retrieved 2008-04-06.

JRH06 Jacobsson, M., Rost, M. and Holmquist, L. E., When media gets wise: Collaborative filtering with mobile media agents. *IUI '06: Proceedings of the 10th International Conference on Intelligent User Interfaces*, 2006.

KKP+03 Korpipää, P., Koskinen, M., Peltola, J., Mäkelä, S.-M. and Seppänen, T., Bayesian approach to sensor-based context awareness. *Personal Ubiquitous Comput.*, 7,2(2003), pages 113–124.

LGP07 Gnu lesser general public license, `http://www.gnu.org/copyleft/lesser.html`, 2007. Retrieved 2008-03-27.

LSY03 Linden, G., Smith, B. and York, J., Amazon.com recommendations: Item-to-item collaborative filtering. *IEEE Internet Computing*, 7,1(2003), pages 76–80.

MAK+97 McKinney, C. H., Antoni, M. H., Kumar, M., Tims, F. C. and McCabe, P. M., Effects of guided imagery and music (gim) therapy on mood and cortisol in healthy adults. *Health Psychology*, 6,4(1997), pages 390–400.

May04 Mayrhofer, R., *An Architecture for Context Prediction*. Ph.D. thesis, Johannes Kepler University of Linz, Austria, October 2004.

Mic08 Microsoft Corporation, Windows media player 11 SDK documentation, `http://msdn2.microsoft.com/en-us/library/bb262657%28VS.85%29.aspx`, 2008. Retrieved 2008-04-13.

Mir07 The 2007 music information retrieval evaluation exchange (MIREX 2007) results overview, `http://music-ir.org/mirex/2007/abs/MIREX2007_overall_results.pdf`, 2007.

MK90 Mori, R. and Kawahara, M., Superdistribution: The concept and the architecture. *Transactions of the IEICE*, E73,7(1990), pages 1133–1146.

MRF03 Mayrhofer, R., Radi, H. and Ferscha, A., Feature extraction in wireless personal and local area networks. *MWCN '03: Proceedings of the 5th International Conference on Mobile and Wireless Communications Networks*. World Scientific, October 2003, pages 195–198.

Nic98 Nichols, D., Implicit ratings and filtering. *Proceedings of 5th DELOS Workshop on Filtering and Collaborative Filtering*. ERCIM, 1998, pages 31–36.

NKL+07 Nurmi, P., Kukkonen, J., Lagerspetz, E., Suomela, J. and Floréen, P., Betelgeuse - a tool for bluetooth data gathering. *BodyNets '07: Proceedings of the 2nd International Conference on Body Area Networks*, 2007.

Öst04 Östergren, M., Sound pryer: Adding value to traffic encounters with streaming audio. *ICEC '04: Proceedings of the 3rd International Conference on Entertainment Computing*, 2004.

Pam06 Pampalk, E., *Computational Models of Music Similarity and their Application to Music Information Retrieval*. Ph.D. thesis, Vienna University of Technology, Faculty of Informatics, 2006.

PB07 Pazzani, M. J. and Billsus, D., Content-based recommendation systems. In *The Adaptive Web*, Brusilovsky, P., Kobsa, A. and Nejdl, W., editors, volume 4321, Springer, 2007, pages 325–341.

PKON04 Patten, C. J. D., Kircher, A., Östlund, J. and Nilsson, L., Using mobile telephones: cognitive workload and attention resource allocation. *Accident Analysis & Prevention*, 36,3(2004), pages 341–350.

RIS+94 Resnick, P., Iacovou, N., Suchak, M., Bergstrom, P. and Riedl, J., Grouplens: an open architecture for collaborative filtering of netnews. *CSCW '94: Proceedings of the ACM conference on Computer supported cooperative work*, New York, NY, USA, 1994, ACM, pages 175–186.

RM06 Reddy, S. and Mascia, J., Lifetrak: music in tune with your life. *HCM '06: Proceedings of the 1st ACM international workshop on Human-centered multimedia*, New York, NY, USA, 2006, ACM Press, pages 25–34.

Sal89 Salton, G., *Automatic text processing: the transformation, analysis, and retrieval of information by computer*. Addison-Wesley Longman Publishing Co., Inc., Boston, MA, USA, 1989.

Sav99 Savan, A., The effect of background music on learning. *Psychology of Music*, 27,2(1999), pages 138–146.

SBG99 Schmidt, A., Beigl, M. and Gellersen, H.-W., There is more to context than location. *Computers and Graphics*, 23,6(1999), pages 893–901. URL `citeseer.ist.psu.edu/schmidt98there.html`.

Sch02 Schmidt, A., *Ubiquitous computing – Computing in context*. Ph.D. thesis, Lancaster University, Computing Department, November 2002.

SFHS07 Schafer, J., Frankowski, D., Herlocker, J. and Sen, S., Collaborative filtering recommender systems. In *The Adaptive Web*, Brusilovsky, P., Kobsa, A. and Nejdl, W., editors, volume 4321, Springer, 2007, pages 291–324.

Slo91 Sloboda, J. A., Music structure and emotional response: Some empirical findings. *Psychology of Music*, 19,2(1991), pages 110–120.

SM86 Salton, G. and McGill, M. J., *Introduction to Modern Information Retrieval*. McGraw-Hill, Inc., New York, NY, USA, 1986.

SM95 Shardanand, U. and Maes, P., Social information filtering: algorithms for automating "word of mouth". *CHI '95: Proceedings of the SIGCHI conference on Human factors in computing systems*, New York, NY, USA, 1995, ACM Press/Addison-Wesley Publishing Co., pages 210–217.

SPUP02 Schein, A. I., Popescul, A., Ungar, L. H. and Pennock, D. M., Methods and metrics for cold-start recommendations. *SIGIR '02: Proceedings of the 25th annual international ACM SIGIR conference on Research and development in information retrieval*, New York, NY, USA, 2002, ACM, pages 253–260.

ST07 Scheible, J. and Tuulos, V., *Mobile Python: Rapid Prototyping of Applications on the Mobile Platform*. Wiley, 2007.

SZ01 Scherer, K. R. and Zentner, M. R., *Emotional effects of music: production rules*. Music and emotion: Theory and research. Oxford University Press, Oxford, 2001.

WMB01 Wallin, N. L., Merker, B. and Brown, S., editors, *The Origins of Music*. MIT Press, 2001.

WPVS05 Wijnalda, G., Pauws, S., Vignoli, F. and Stuckenschmidt, H., A personalized music system for motivation in sport performance. *IEEE Pervasive Computing*, 4,3(2005), pages 26–32.

YGK+06 Yoshii, K., Goto, M., Komatani, K., Ogata, T. and Okuno, H. G., Hybrid collaborative and content-based music recommendation using probabilistic model with latent user preferences. *ISMIR '06: Proceedings of the 7th International Conference on Music Information Retrieval*, 2006, pages 296–301.