DIPLOMARBEIT

Innovative User Interfaces for accessing Music on Mobile Devices

Ausgeführt am
Institut für Softwaretechnik und Interaktive Systeme
der Technischen Universität Wien

unter der Anleitung von Ao.Univ.Prof. Dr. Andreas Rauber

durch

Peter Hlavac
Deublergasse 37, 1210 Wien
Matr. Nr. 9625925

Wien, März 2007

Eidesstattliche Erklärung

Ich erkläre an Eides statt, dass ich die vorliegende Arbeit selbständig und ohne fremde Hilfe verfasst, andere als die angegebenen Quellen nicht benützt und die den benutzten Quellen wörtlich oder inhaltlich entnommenen Stellen als solche kenntlich gemacht habe.

Danksagung

Diese Arbeit ist meiner Oma gewidmet, deren größter Wunsch ist, meinen Abschluß miterleben zu dürfen.

Danke Mama, daß Du seit es mich gibt, einfach immer für mich da gewesen bist.

Speziellen Dank an meinen Freund Jörg, der mich stetig mit seinem weitsichtigen Blick von Oben und zahlreichen Motivationsschüben durch die Endphase meines Studiums begleitet hat.

Zusammenfassung

In dieser Arbeit werden unterschiedliche Visualisierungsarten von Musiksammlungen auf mobilen Endgeräten vorgestellt. Weiters werden innovative Ideen zum Navigieren durch Musiksammlungen präsentiert. Um die angeführten Konzepte auf mobilen Endgeräten zu demonstrieren, wurde der Prototyp **MobileSOM** (Mobile Selection of Music) entwickelt. Der Prototyp basiert auf einem Grid-Unit Verfahren, welches eine Musikbibliothek als 2-dimensionale Musiklandkarte (*Grid*) in unterschiedlichen Visualisierungen darstellt, wobei Musikstücke mit ähnlichen Eigenschaften in Feldern (*Units*) zusammengefasst werden. MobileSOM verfügt über ein intuitives User Interface, das dem Benutzer ein einfaches Auswählen von beliebigen Musiktiteln auf der Landkarte ermöglicht und diese in einer Playlist abspeichert. Die Lieder können entweder direkt vom Gerät selbst oder von einem im Netzwerk verfügbaren Server abgespielt werden. Außerdem kann das Gerät in Kombination mit MobileSOM als Fernbedienung benutzt werden. Dabei werden die ausgewählten Songs auf einem anderen Gerät abgespielt und die Wiedergabe durch die Software ferngesteuert. Abhängig von den Ausstattung des mobilen Endgerätes kann der Prototyp Lieder in verschiedenen Audioformaten wie mp3, wav, amr, usw. abspielen. MobileSOM wurde nicht dafür konzipiert, um punktgenau einen bestimmten Titel zu entdecken. Es gibt keine speziellen Suchmechanismen um genau einen vom Benutzer gewünschten Song zu finden. Vielmehr geht es darum, den Benutzer bei der Erstellung von Playlisten, abgestimmt auf dessen Stimmungslage oder Hörverhalten, zu unterstützen.

MobileSOM wurde in Java Micro Edition (J2ME) programmiert und ist somit einem breiten horizontalen Markt von mobilen Endgeräten zugänglich. Der Prototyp ist der erste verfügbare Audioplayer in J2ME, mit dem das Erstellen von Playlisten auf einer Musiklandkarte möglich ist. Zur Demonstration wurden zum Zeitpunkt der Implementierung die modernsten mobilen Endgeräte bestimmter Hersteller ausgewählt, um maximale Ressourcen für die rechenintensive Multimedia-Anwendung bereitzustellen. Trotz erfolgreicher Tests liefen auf Grund von eingeschränkter Leistungsfähigkeit der Hardware nicht immer alle Bestandteile der Software fließend. Der Speicher für mobile Anwendungen in J2ME ist sehr begrenzt, die Darstellung des User Interaces zum Teil sehr schleppend, die Wiedergabe von mp3 Dateien in Java wird nur auf wenigen Geräten unterstützt und das Streamen von Musikdateien in hoher Qualität über das Netzwerk ist oft mit längeren Verzögerungen verbunden. Bei den nächsten Generationen von mobilen Endgeräten werden die

erwähnten Schwachstellen der Vergangenheit angehören. Der in dieser Arbeit präsentierte Prototyp bietet somit Anreiz für die Umsetzung der vorgestellten Konzepte in kommerziellen Musikanwendungen in der nahen Zukunft.

Abstract

The thesis introduces a framework for displaying music libraries in different visualisation styles on mobile devices. Furthermore innovative browsing techniques for navigating through music collections are presented. To demonstrate the concepts for exploring music on a mobile device the prototype **MobileSOM** (Mobile Selection of Music) was implemented. The prototype offers different visualisation styles of music libraries based on the Grid-Unit concept which displays a music collection on a two-dimensional map (*Grid*) where pieces of music with similar properties are grouped into *Units*. Moreover MobileSOM has an intuitive user interface that enables a simple way of selecting music titles from the Grid and adding them to a playlist. Playlists generated in this manner are played on a mobile device or streamed from a web server. The device can also be used as a remote control, i.e. instead of playing a piece of music, the software triggers another device to play the song. Depending on the audio capabilities of the device the prototype plays songs in various audio formats like mp3, wav, amr, etc. MobileSOM is not designed for finding a specific piece of music. There are no special search mechanisms implemented for locating exactly the one song that a user has in his mind. MobileSOM assists in generating playlists that fit into a specific mood or style and encourages the user to explore unknown music content rather than to look for the latest Britney Spears hit.

MobileSOM is programmed in Java Micro Edition (J2ME) making the program available to a wide horizontal market of mobile devices and it is the first J2ME audio player ever that implements the concept of generating playlists using a two-dimensional music map. For demonstration purposes a small set of state-of-the-art devices were carefully selected from various manufacturers to run a slimmed version of MobileSOM. Testing the prototype on real world devices showed that the presented concepts are practicable however, not all implemented features perform smoothly because of hardware limitations: The maximum allowed size for a J2ME application running on a mobile device is very small. The graphical computation power on most devices is weak, only a few devices support an implementation of the JVM that plays back mp3 files and last but not least the network speed for streaming music in high quality needs to be increased.
MobileSOM has not yet reached a level, which would suggest its immedeate commercial usage. However, next generation devices will overcome the technological bottle neck so that this work will be a great inspiration for commercial music applications running on mobile devices in the near future.

Contents

Chapter 1
Introduction

1.1 Mobile Music Market

In the age of digital music distribution getting a specific piece of music is easier than ever. Today's commercial music download platforms like iTunes[1] or the European counterpart OD2[2] offer catalogs with more than 1.5 million songs. The number of legal downloads of single audio tracks increased in Europe in 2006 from 65 to 111 millions, which is an increase by 80% [IFP07]. Thanks to the growing popularity of 3G networks the mobile market will be a main distribution channel not only for ring tones but also for full track downloads. Increasing bandwidth and lower data transfer costs encourage users to access digital content over their mobile phones. Napster[3] offers over 3 Million tracks any time and anywhere for consumers to enjoy music on the PC, their mobile phone or on an MP3 player. For a monthly flat rate Napster customers can stream or download music online via fixed or mobile devices and reproduce their songs without limitations, as long as the monthly subscription is maintained. Mobile Operators extend their portfolio with music services like the Austrian network operator ONE who launched the "Ladezone" [4] in 2006. Beside ring tones, games and other styling assets users can purchase and download music from a repertoire with more than hundred thousand songs directly to their mobiles. More and more people use their mobile devices for listening to music, which is confirmed by the number of sold portable music players that totalled around 120 million in 2006, an increase of 43 per cent on the previous year[IFP07]. To sum up: The mobile music market is growing!

1.2 Consuming Music

For desktop machines many services for consuming music from the Internet have evolved. Online radio stations like Pandora[5] or last.fm[6], commercial

[1] http://www.apple.com/itunes/
[2] http://www.od2.com
[3] http://www.napster.com/napstermobile/
[4] http://www.ladezone.at
[5] http://www.pandora.com
[6] http://www.last.fm/

download platforms offering different styles of music like Beatport[7] or online shops like the Black Market[8] offer consumers easy access to their favourite music. In case a user wants to listen to his music elsewhere than on his PC, he might transfer his library to a mobile device. Consuming music on mobile devices is less convenient because they have fewer resources compared to average desktop systems. Displays are much smaller and the lack of a keyboard makes it difficult to find a specific song. User interfaces and program logics must be adapted or even redesigned for those devices. The focus of the work presented is on graphical user interfaces and navigation opportunities that enhance the access to music on mobile devices. To demonstrate new ways for exploring music, the prototype **MobileSOM** (Mobile Selection of Music) was implemented. The prototype offers a framework for using different visualisation styles of music libraries based on the Grid-Unit concept which displays a music collection on a two-dimensional map where pieces of music with similar properties are grouped into units. Moreover MobileSOM has an intuitive user interface that simplifies selecting music titles from the map and adding them to a playlist. Playlists generated in this manner can then be played on the mobile device or sent to an external player. The prototype is programmed in J2ME making the program available to a wide horizontal market of mobile devices.

1.3 Scope and Overview of the Thesis

The main contribution of this thesis is the design and implementation of the prototype MobileSOM. In the design phase intensive research has been carried out to find a group of mobile devices of interest, setting up a suitable developing environment and how to realize different visualization concepts on mobile devices. The thesis introduces existing approaches to exploring music libraries both on desktop machines and mobile devices in *Chapter 2*.

Chapter 3 discusses the evolution of mobile devices and their features. Furthermore the state-of-the-art devices used for the demonstration of the prototype MobileSOM are presented.

In *Chapter 4* concepts for exploring and displaying music content on mobile devices with respect to the application MobileSOM are shown.

Chapter 5 gives an overview of the integrated development environment and shows how to design a mobile application in general.

The central chapter of the thesis, *Chapter 6*, provides a detailed description of the software architecture, followed by an installation guide for deploy-

[7] https://www.beatport.com/
[8] http://www.soulseduction.com/

ing and running the prototype on real world devices. The user interface and the interaction possibilities within the software are explained. At the end of the chapter practical experiences of running MobileSOM on real world devices and ideas for extending the software with new features are discussed.

Finally, in *Chapter 7*, the work presented in the thesis is summarized and an outlook for using MobileSOM in commercial applications is given.

Chapter 2
Related Work

The work has originated in the project SOM-enhanced JukeBox (SOMeJB) [RPM03], developed by the Institute of Software Technology and Interactive Systems at the Vienna University of Technology. The SOMeJB is an approach to automatically create an organization of a music archive following the sound similarity of the music files. More specifically, characteristics of frequency spectra are extracted and transformed according to psychoacoustic models. Resulting psychoacoustic rhythm patterns are further organized using an unsupervised neural network, the Growing Hierarchical Self-Organizing Map (GH SOM) [RMD02]. Particularly, a Self-Organizing Map is employed to create a map of a musical archive, where pieces of music with similar sound characteristics are organized next to each other on the two-dimensional map display. Locating a piece of music on the map, leaving a user with related music next to it, allows intuitive exploration of a music archive. Different visualization techniques for a map can be found in Islands of Music (IoM) [Pam01]. The applications PlaySOM [DNR05] and PocketSOMPlayer [NDR05] enable users to browse music collections, select tracks, export playlists as well as listen to the selected songs. The PlaySOM presents a rich interface, offering different selection models, a range of visualizations, advanced playlist refinement, export functionality to external player devices and a basic playback mode of selected songs. The PocketSOMPlayer, on the other hand, offers a slim version of the desktop application, optimized for mobile devices. MobileSOM is an enhanced version of the PocketSOMPlayer with more sophisticated features and is the first SOM-based audio player entirely programmed in J2ME.

Other applications that inspired the design of MobileSOM were ZuiScat[BR05] and CoMIRVA[Sch06]. ZuiScat is a visualization concept for querying large information spaces on Personal Digital Assistants (PDAs). Retrieval results are presented in a dynamic scatter plot, which is enhanced by geometric and semantic zoom techniques to provide smooth transitions from abstract visual encodings to data content. CoMIRVA[1] is a framework implemented in Java for hosting various algorithms concerning music, multimedia, information retrieval, information visualization, and data mining.

[1] http://www.cp.jku.at/people/schedl/Research/Development/CoMIRVA/webpage/CoMIRVA.html

Figure 2.1: iPhone: Browsing through albums using the RSVP technique

Figure 2.2: LyricShow Player: A audio player programmed in J2ME showing the lyrics while playing the song

The latest invention from Apple, the iPhone[2], uses a smart technique called Rapid Serial Visual Representation (RSVP)[dBS00] for browsing through a user's entire music collection on the device (see Figure 2.1). LyricShow Player[3] is an audio player programmed in J2ME that runs song lyrics synchronously to pieces of music while played.

[2]http://www.apple.com/iphone/
[3]http://www.mobile-mir.com/en/LyricShow.php

Chapter 3
Mobile Devices

Before discussing sophisticated user interfaces for exploring music on mobile devices, the first question to answer is: What is a mobile device? Let's start with some definitions that straiten the imagination of what a mobile device is and what it is not or to put it differently: What kind of mobile devices do we need to run innovative music browsing software on? At the end of this chapter the reader will understand the prerequisites a device must be capable of in order to put into practice the concepts that will be presented in the thesis.

In general, a mobile device is a light-weight technical accessory a user can carry and interact with. A primitive pocket calculator or a wristwatch already fits this definition. For more complex user interaction a mobile device provides physical elements for interaction like a keyboard, a pointing device for touch sensitive devices, a control pad or a wheel for pointing to or selecting from the information displayed. Less conventionally, some devices have position and movement sensors. Even squeezing a device is used for interaction [HFG+98]. To give feedback to the user about its own state, the device it is equipped with visual or aural output capabilities like a display, loudspeakers or mechanical powered functions like vibration. Figure 3.1 on page 16 groups mobile devices into three categories: PDAs, communication- and music devices. This is only a small subset of groups of mobile devices available but puts the focus on the devices of interest to run music browsing software on. Every category shows four representatives (a-d) chronologically ordered. After discussing the evolution and features of mobile devices the reader knows about the actual state-of-the-art devices and which devices according to the following features are suitable for the prototype MobileSOM presented in this work:

- Interaction
- Memory
- Connectivity
- Synchronization
- Customization

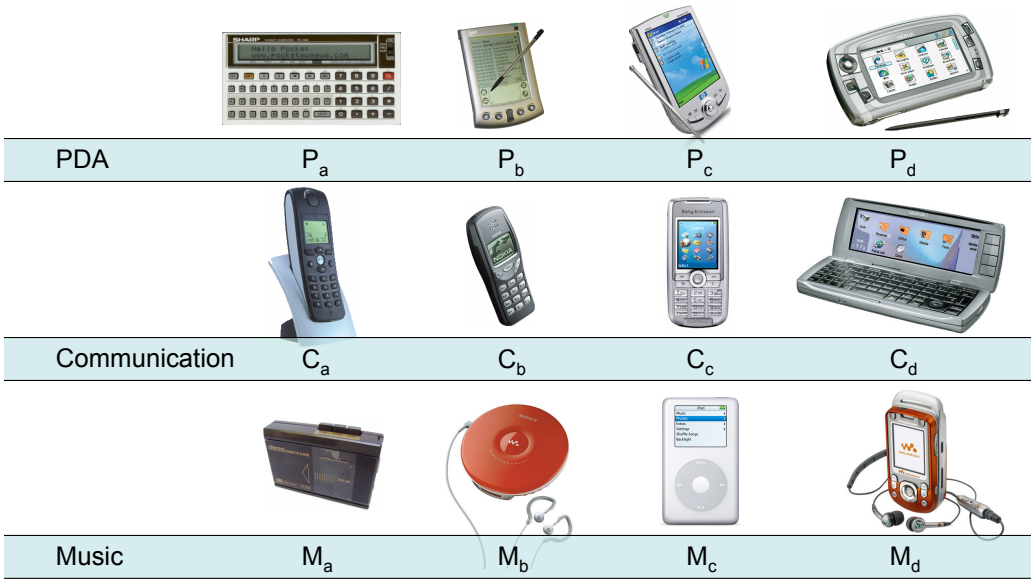

Figure 3.1: Mobile Devices divided in three Categories: PDAs, Communication- and Music Devices

3.1 Personal Digital Assistants (PDAs)

3.1.1 History

The term "personal digital assistant" got popular at the Consumer Electronics Show in Las Vegas, Nevada in 1992 used by the Apple Computer CEO John Sculley [Kob05]. In fact, PDA forerunners were available as early as the mid-1970s; first as very advanced calculators, see Figure 3.1-P_a, then as electronic organizers like palmtops or pocket PC's in Figure 3.1-$P_{b,c}$ and today as pda-mobile-phone hybrids like the Nokia 7710 shown in Figure 3.1-P_d.

3.1.2 Features

Interaction

PDAs are equipped with touch screens for user interaction, some buttons that are usually reserved for shortcuts to primarily used programs and a detachable stylus. Interaction is done by tapping the stylus on the screen for example to click on buttons and menu items or dragging the stylus to highlight text. For entering text into a PDA one possibility is to use a virtual keyboard that is shown on the screen. Text input is done by tapping the letters. Another way is to use text recognition, where letters or words are

drawn on the touch screen, and then translated to letters filled in the current textbox. Although many years of research and development have been done to improve word recognition, this input method is still a time consuming task for the user and it tends to be rather inaccurate [Mas98]. Some PDAs, like for example the BlackBerry, have a full keyboard and scroll wheels to facilitate data entry and navigation in addition to the input methods described above. Another way for interacting with a stylus on a touch screen is to use gestures. Gestures are small movements that are used as shortcuts to execute customized functions. For example: Dragging the stylus from right to the left is interpreted for deleting a letter. There are plenty of possible gestures that can be defined by the user and assigned to specific actions [Rub91] [MC02].

Memory

PDAs use internal memory for running the operating system and executing programmes. For personal data that is too big to be kept in internal memory, like photo albums, music files or even video collections, PDAs provide slots for external memory cards with capacities currently up to 8GB.

Connectivity

All PDA's have at least an infrared port (IrDA) for connectivity. This ensures communication between different PDAs or between a PDA and a computer with an IrDA interface. Today's PDAs also have a Bluetooth port or a WiFi interface for faster wireless connectivity.

Synchronization

Synchronization is used to keep personal data like contacts, emails and in context to this work music libraries on both the mobile device and the user's desktop machine up to date. Because of time consuming text input methods, slower connectivity and less memory capabilities on PDAs most data is created at the host computer and then transferred to the PDA. Data created on the PDA when exchanging contacts or sharing files between mobile devices is copied back to the PC during the synchronization process. The PC acts as a backup storage in case the device gets lost or corrupted.

Customization

PDAs can be customized by installing third party software or connecting external gadgets like foldable keyboards for quicker text input or GPS devices to use a PDA as a navigation instrument.

3.2 Mobile Phones

3.2.1 History

The term "mobile" or "cellular telephone" is used for portable electronic devices designed primarily for personal telecommunications over long distances, see Figure 3.1-C$_{b,c}$, unlike "cordless telephones" which communicate with a base station connected to a fixed telephone landline, see Figure 3.1-C$_a$. Martin Cooper is one of the most important inventors of the cell phone. He made the first call on his cell phone in 1973 [WIK05]. Until the mid to late 1980s, most mobile phones were installed in vehicles as car phones because they where to big to be carried permanently. With the advance of miniaturization, current mobile phones are now used as handhelds. Moreover, modern cell phones merge with PDAs as pda-mobile-phone hybrids (called Smartphones) like the Nokia 9500 Communicator in Figure 3.1-C$_d$.

3.2.2 Features

In addition to the standard voice function of a telephone, a mobile phone supports additional services such as SMS for text messaging and MMS for sending and receiving photos and video. They can also send and receive binary data and faxes, access WAP services, and provide full Internet access using technologies such as GPRS. Most current models have a built-in digital camera plus sound and video recording capabilities. GPS receivers are starting to appear integrated or connected via bluetooth to cell phones. Push to talk is a feature that allows a person to talk to another one by holding a speech button similar to walkie-talkies.

Interaction

At the time when mobile phones were used for voice communication only, a small black and white display and a numeric keyboard for dialling numbers was sufficient. For more complex features, like browsing through web pages, a bigger display is required. For example the Nokia 7710 is equipped with a 3.5 inch TFT wide screen with 65.536 colours and a resolution of 640 x 320 pixels. Because the popularity of writing emails on mobile devices increased, some phones also feature full alphanumeric keyboards, such as the Nokia 6820 or the BlackBerry. Another interaction feature is voice recognition that can be used to command the device to dial a number or to inform the user about the next appointments.

Memory

With the significant enhancement of the camera capability of mobile phones the memory capacities of mobile devices have to increase as well. The Nokia N90 has a 2M pixel camera and can record video at 352x288 pixels and 15 frames per second. The Nokia N93 is reported to provide DVD quality video at 30 frames per second with an internal memory of 50 MB and an external slot for cards with capacities up to 2GB [Nok].

Connectivity

Besides sharing contact information over infrared and Bluetooth mobile phones are now heavily used for data communications such as SMS messages, browsing web sites or even streaming audio and video files. Connection speed is based on network support. The data part of the GSM protocol is called GPRS. A significant number of models already support third-generation (3G) communications UMTS[1] - generally a downlink of up to 384kb/s and an uplink of up to 64kb/s. Recent models such as the Nokia 6680 and the Nokia N90 have access to the Web via a free download of the Opera browser.

Synchronization

Like PDAs, Mobile Phones can synchronize their contacts, messages and dates with a desktop machine. For example Nokia phones use the software Nokia PC Suite[2] to connect with the PC. The application offers synchronization and backup facilities, direct access to the file system on the device, composing ring tones and much more.

Customization

A big milestone in the mobile phone history is the implementation of a Java Virtual Machine (JVM) on mobile devices. The Java Platform, Micro Edition (Java ME or J2ME)[3] makes it possible to execute third party software programmed in Java on a mobile device. Other customisation options are styling features aimed toward personalisation, such as downloadable or self composed ring tones, logos and interchangeable covers.

[1] http://en.wikipedia.org/wiki/Universal_Mobile_Telecommunications_System
[2] http://www.nokia.de/de/service/software/pc_suite/114940.html
[3] http://java.sun.com/javame/index.jsp

3.3 Portable Music Players

3.3.1 History

Portable music players are handheld devices that playback any desired music. In the 1980s Sony invented the Walkman, a device that was able to play music from compact cassettes with durations up to two hours (see Figure 3.1-M_a). Cassettes were replaced by Compact Discs (CD) and the player for that medium was called Discman (see Figure 3.1-M_b). The advantage of a CD is that the quality of the digitally recorded music stays the same as long as the CD is not damaged unlike analogue sound material that gets worse the more often it is played or the older it gets. Derivatives of the Discman were the MiniDisc Players. They use a compressed audio format and the medium is physically a little smaller than a CD. Newer players that play digital audio files, like the iPod in Figure 3.1-M_c, are called digital audio players (DAP) or colloquially Mp3 Players. They have much more capacity and more interaction features than the generations of audio players had before. Today cellphone-walkman-hybrids like the Sony Ericsson W550i Walkman in Figure 3.1-M_d are even able to stream music from different sources over the Internet.

3.3.2 Features

Interaction

Every portable music player has buttons for navigating through its music content. Cassette Players were able to start or stop the playback at any time and fast forward the tape. With Discmans it is even possible to skip whole audio tracks with one click. Loudness is controlled with buttons or a wheel by the user. Some players even have equalizers where given frequencies can be manipulated with sliders in real-time. Most Discmans are equipped with a small LCD that shows the current time of the song playing, the track number of the CD, the battery status and much more. To navigate through big music libraries Apple invented the "iPod wheel" [App], where it is easy to skip a big number of tracks by turning the wheel. The wheel is also used for fast forwarding within a track or to control the volume. Bigger displays of portable music players are able to show more information like metadata for songs (e.g. title, artist, and album) or a directory structure ordered by artist name or genre where users can step through different folders with primitive cursor buttons. The high resolution colour display of an iPod even shows the album cover of the currently played song if available.

Memory

Compact cassettes had a capacity up to two hours and compact discs can hold up to 90 minutes of audio material recorded in 44khz/16bit stereo. With the invention of the MPEG-1 Audio Layer 3[Bra03], more commonly referred to as mp3, in the beginning of the 1990s it was possible to compress music from ten CDs into a single one with almost no audible loss. The first mp3 players with storage of only 64MB where able to play back a complete CD. Modern players are able store up to 80GBs which is equal to a playtime of about 20.000 Songs (=one and a half month non-stop music(!)). However, the trend will be not to have overblown amounts of memory - it goes the other way around. Audio players of the future will have no or very little music on them at all. Like the name says they will play audio but the source can be anywhere, like on a private desktop PC or on the favourite artist's webpage. Music will be dynamically streamed to the device so there is no need to have unlimited memory capabilities [Eri]. Streaming is a method for making music, video, radio and other multimedia available in real-time or near real-time, over different types of networks. The data in the file is split into small packets that are sent in a continuous flow, or a "stream", to the end user's computer or mobile phone. The user can begin listening to the content in the first packets, while the rest are being transferred. There is a short delay at the start to allow the client to buffer a small amount of data. This buffer makes it possible for the client to play the stream without interruption, even if the rate of received data varies slightly. In the case of streaming, the audio file is not stored on the user's device, so in order to listen again, the user will have to reconnect to the streaming server and initiate another streaming session.

Connectivity

Digital audio players are connected over USB2.0 or Firewire to the PC. This theoretically allows transfer rates up to 480 MBit/s. In practice an album with approximate 80MB needs about 30 seconds to be copied to a device. Microsoft's latest DAP, called Zune[4], released in November 2006, has IEEE 802.11[5] networking built in for wireless connectivity with transfer rates up to 54 MBit/s [Mic07]. In practice sending a piece of music with about 4MB over the air takes about 2 seconds. Sony Ericsson W550i Walkman is able to stream music files over GPRS with a pre-delay of about 20 seconds which is the time a song needs to be buffered in the device first.

[4]http://www.zune.net

[5]http://en.wikipedia.org/wiki/IEEE_802.11

Synchronization

Synchronization between a digital audio player and a desktop PC is usually a one way process. The entire music archive or a subset is transferred from the PC to the player. If new songs are purchased from the Internet or ripped from CDs, they are automatically synchronized with the player. The iPod uses a docking station connected via FireWire to the PC and the software iTunes [6] that manages all transfers from and to the device. Microsoft's player Zune has wireless synchronization facilities. It is possible to share songs between Zune players with some restrictions, for example a shared song can be listened for three times only. Using synchronization software like the Windows Media Player has one big restriction called digital rights management (DRM[7]). Pieces of music are using a protected file format called WMA-DRM and they can be played only on the PC where they were downloaded or on the portable player they were transferred to.

Customization

The most obvious customization of any music player is of course the music itself. But there are plenty other customization possibilities, like designing an own case for an iPod seen on ifrogz[8], connecting special headphones with extra bass enhancement, install third party software for more convenient user interaction or even replace the existing operating system (e.g. iPodLinux[9]).

3.4 Summary

This chapter described three different types of mobile devices grouped into PDAs, communication- and music devices. The newer devices are, the more difficult it is to assign them to a specific group. For example the Nokia 7710 fits in any of these groups because it may act as a portable music player, has all functionality of a PDA and it can be used as a mobile phone as well. These devices are called **Smartphones** and the focus of this work will be in such all-in-one devices for the following reasons:

[6] http://www.apple.com/en/itunes/
[7] http://www.microsoft.com/windows/windowsmedia/forpros/drm/default.mspx
[8] http://ifrogz.com/
[9] http://ipodlinux.org/

3.4.1 Interaction

The main limiting factors of user interaction are the screen size, the lack of a keyboard, processing power and connection speed. With Smartphones the screen resolution has become bigger and the computation power got much stronger. The newer generation of phone CPUs run at over 400 MHz and their displays are rich of colours and support resolutions up to 640 x 320. The built-in touch screen allows complex interaction tasks like drawing an object on the screen or dragging elements from one position to another.

3.4.2 Memory

Smartphones have enough memory capabilities (up to 8GB) to store music or even video collections on them, which makes them an optimal portable music player.

3.4.3 Connectivity

Even if memory capacity is limited the ability to connect to the Internet using UMTS enables Smartphones to stream or download new music from the Internet with a downlink of up to 384kb/s.

3.4.4 Synchronization

Music downloaded on the mobile device or on the PC can easily be exchanged with simple file synchronisation mechanisms.

3.4.5 Customization

The support of a JAVA runtime environment (J2ME) on most Smartphones makes it easy to experiment with ideas for user interaction. Implementing concepts for user interfaces is a straight forward process and programming in J2ME is very similar to programming JAVA applets or JAVA swing components. A drawback for rich multimedia applications is the fact that applications that are accessing the file system need a fee required certificate and playing music files, no matter if they come from internal memory or expansion cards, is coupled with a long pre-delay. Device specific code works more efficient but to get a look and feel about the prototype, the performance of J2ME is sufficient. J2ME is called to be a device independent programming language. Why this is not the case will be discussed in detail in section 6.7.

More powerful Smartphones of the next generations will overcome those bottle necks so that this work will be a great inspiration for commercial music applications on mobile devices in the near future.

3.4.6 Devices selected for MobileSOM Evaluation

After a precise research about mobile devices available today, studying their features and experimenting with a few of them the following devices were used and tested successfully with the prototype MobileSOM:

- Nokia 7710
- Sony Ericsson M600
- Qtek 9100
- Benq P50

All those Smartphones are able to run J2ME applications. They support multimedia facilities, touch screen interaction and high speed Internet access. Detailed Information about the devices can be found in Appendix C.

Chapter 4
Concepts for Exploring Music Libraries

In this chapter concepts for exploring and displaying music content on mobile devices with respect to the application MobileSOM are presented. The first section deals with music libraries in general and how metadata from music files are generated. In the following section the Grid-Unit concept is introduced and the reader is given an idea of how pieces of music are organised on a 2-dimensional music map. Different visualisation styles as well as methods for arranging units on a grid are presentend in section 4.3. Finally, an approach for generating playlists using the Grid-Unit Concept on a mobile device is discussed.

4.1 Music Libraries

A music library is a collection of pieces of music that is typically organised in an ordinary folder structure on a device. Most popular file formats that compress audio data are .mp3, .wma, .aac in contrast to uncompressed audio data like .wav or .aif files. Some of these formats use copy protection mechanisms better known under the term "Digital Rights Management" (DRM)[Med01] or less restrictive techniques like watermarking an audio file[Arn99]. The more music libraries grow, the easier a user loses track of the location for specific pieces of music and the task finding music concerning a user's taste gets more complex. Different approaches for accessing music more enjoyably have evolved and are discussed in the following sections.

4.1.1 Tagging

Digital audio files may contain, in addition to the audio signal, related text information (e.g. lyrics) or graphical data (e.g. an album cover). Typical text attributes are for example the song title, the artist name, the album name, the release date, the genre, etc. The process of including information other than sound into these digital audio files is commonly referred to as "tagging". The original standard for tagging digital files was developed in mid-1990s by Eric Kemp [1] and he coined the term ID3 which simply means "IDentify an MP3". A user can find pieces of music by additionally adding those

[1] http://www.id3.org/

tags in a search request instead of searching for filenames only. However, tagging is done by humans, who may describe the same piece of music with inconsistent data. Ambiguous tags like genre [AP03] may be associated with different styles of music by different people. The spelling of artists may differ as well, particularly if a name contains special characters, like ',# or &. If users are searching for music from the genre "alternative rock" in a foreign music library they might get different results than they are associating with this category.

4.1.2 Automatic Metadata Generation

To overcome the ambiguity of tagging the same songs with different values an approach called feature extraction is used to automatically generate objective metadata without the need of human input.

Feature Extraction

The audio signal of a music files are analysed by the computer to automatically extract measurably attributes called **features** which are stored in a feature vector. The more features are extracted the potentially more accurate a piece of music is described. Figure 4.1 shows two songs in a simplified 2-dimensional feature space using the features loudness and tempo. Songs that have about the same tempo and loudness are mapped closer to each other than the ones that have different values. In research projects [MUNS05],[RF01] feature spaces are using a much higher dimensionality ranging from 20 to 4352 dimensions.

The Distance

Now that songs are arranged in an n-dimensional feature space it is possible to find "similar" pieces of music to each song. This is typically done by computing the Euclidian distance between two feature vectors. The resulting value gives an indication about the similarity of two songs; the smaller the distance is, the more the songs are related to each other. This distance can then be used to generate playlists or help users to explore music libraries more intuitively. Soundscout[2] is a demonstrator for browsing through a music library containing about 60.000 songs. After a base song is chosen by the user, soundscout recommends songs ordered by their smallest distance to the given song.

[2]http://soundscout.researchstudio.at/

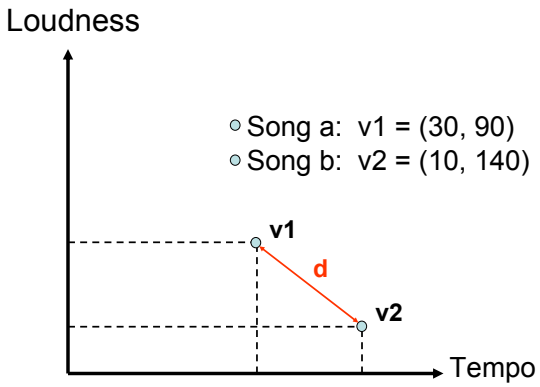

Figure 4.1: Two songs represented as feature vectors (v1, v2) in a 2-dimensional feature space with the features tempo on the x-axis and loudness on the y-axis. The distance d is euclidian distance between the two vectors.

4.1.3 Collaborative Filtering

A recent technique for creating metadata is called collaborative filtering which is a method of making automatic predictions (filtering) about the interests of a user by collecting taste information from many users (collaborating). The underlying assumption of the approach is that users who agreed in the past tend to agree again in the future. A collaborative filtering or recommendation system for music tastes could make predictions about which music a user should like given a partial list of that user's tastes (likes or dislikes). Pandora [3] and last.fm[4] which are interactive radio stations on the web, are using this technique to bring personalized music to the people.

4.2 The Grid-Unit Concept

A simple way of displaying content on a graphical user interface is to use a *Map Grid* that is divided into a set of *Units*. Every Unit contains one or more *Map Items* but might be empty as well (see Figure 4.2). A Map Item is identified by its unique id and is described by attributes like its name, size or creation date. The idea of using Units is to group items with similar properties together. In case Map Items are pieces of music a Unit either

- contains songs from the same artist

[3] http://www.pandora.com/
[4] http://www.last.fm/

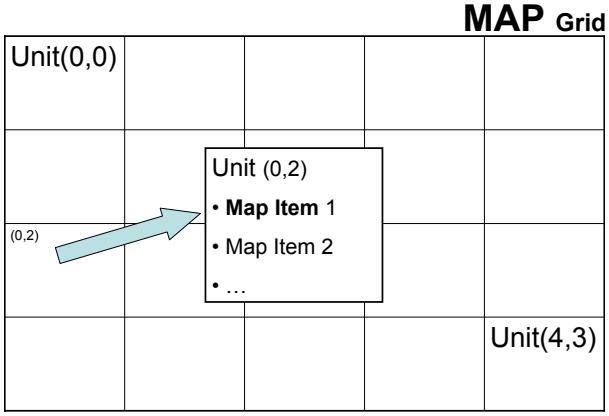

Figure 4.2: A MAP Grid with 20 Units subdivided into five columns with four rows displaying two Map Items associated with Unit(0,2)

- is a placeholder for an album or a sampler
- stands for a specific genre
- groups similar sounding pieces of music together
- is an abstraction for a specific mood.
- or is manually filled with items

Depending on the metaphor that is used for the Units the number of items associated with a Unit differs. For example a Unit containing songs from a specific genre like "Pop" might has thousands of items in it where as a Unit that is a placeholder for a newcomer artist might only include his or her debut song.

4.3 Visualizing the Grid

In the previous section the idea of organising items on a grid with units that contain items with common properties was presented. Now some ideas for visualizing a grid and its units are given.

4.3.1 Plain Grid

In Figure 4.3(a) every unit is displayed as a box showing the number of the items contained in that unit. The plain grid shows no other information about a unit's content and is only used as an additional layer to be combined with different visualization techniques.

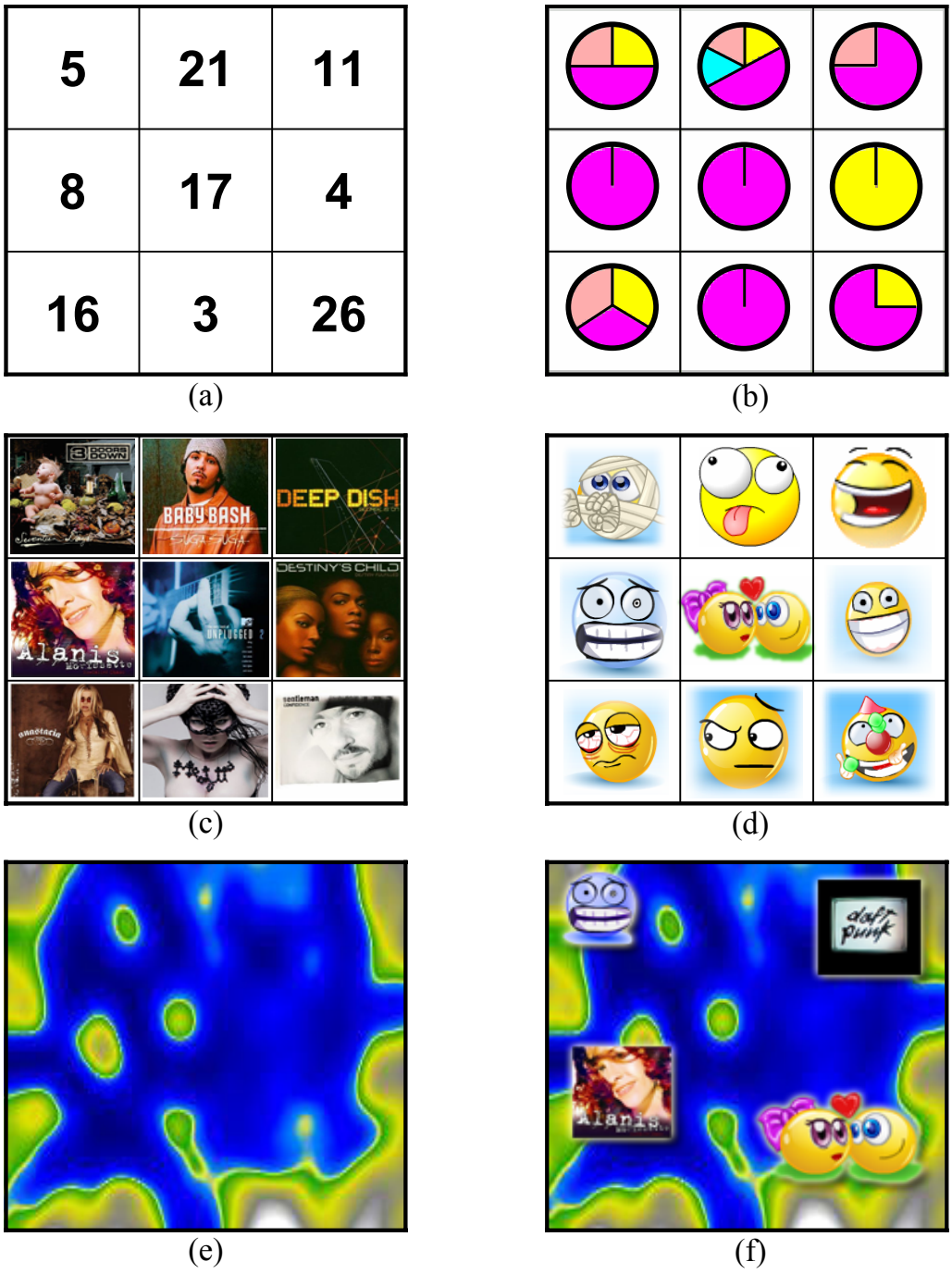

Figure 4.3: Visualizing the Grid

4.3.2 Pie Chart

A visualization technique for describing the content of a unit is to display a pie chart on it. The diagram groups the items of a unit by a specific attribute like genre (e.g. Pop, Rock ...) or realease decade (80tes, 90ties ...). In Figure 4.3(b) a grid with nine pie charts is shown. We assume that every colour stands for a specific genre, like purple for pop, pink for rock, yellow for alternative and cyan for classics. As the reader can see most of the units contain songs from the genre pop. Above the center unit, there is a unit containing all defined genres in the following ratio: 50% of the unit are pop songs and the other occurring genres are euqi-distributed with 16,67% per category.

4.3.3 Album Covers

An intuitive way to visualize the grid is to display an album cover for each unit like shown in Figure 4.3(c). The underlying pieces of music are either all from the same album or the cover belongs to a song that is a representative prototype of that unit. A prototype is chosen randomly or the average feature vector (afv) of all songs in that unit is computed and the piece of music with the smallest distance to the afv is taken.

4.3.4 Metaphoric Symbols

A subjective way to describe a unit is with user defined icons. For example a unit containing pieces of music with rock elements may use an icon with a guitar on it. The visualization technique is also suitable for mapping music content to moods and situations. Many case studies in this sector have been done like [LO03] or the EmoMusic project [Bau06][BR06] which is about classifying music according to emotions (or the mood of the listener). The emotions used are fear, hostility, guilt, sadness, joviality, self-assurance, attentiveness, shyness, fatigue, serenity, and surprise. In Figure 4.3(d) emoticons [5] are used to represent the content of a unit. Every unit is associated to a specific mood or situation. For example the center unit may contains pieces of music for romance and love or the angry looking emoticon below is a place holder for aggressive music. The mapping of songs to a unit is not objective because music associated to a mood differs from person to person. Some people listen to heavy metal music to wake up others use this type of music to get relaxed.

[5] http://en.wikipedia.org/wiki/Emoticon

4.3.5 Self Organising Map

An approach to automatically assign music to units is to apply a Self Organising Map (SOM). A SOM is an unsupervised learning algorithm that is used to project high dimensional data points on a 2-dimensional map. The high dimensional data points are extracted feature vectors from music files (see Section 4.1.2) where each vector is assigned to a specific unit in the 2-dimensional grid. The algorithm causes similar songs to be grouped in units close together. In order to display the resulting map in the software MobileSOM some pre-processing steps have to be done on a desktop machine first. To define how the songs are related to each other regarding their similarities SOMeJB [RPM03] is used to analyse and process the audio files in a way that a computer is able to distinguish between different kinds of music. Features like timbre, loudness and rhythm are extracted by computing the power spectrum of the audio signal to obtain loudness values for specified frequency bands over a certain time period. For a detailed description on feature extraction see [RPM02]. Those features are used to describe a piece of music in a mathematical representation and are stored in a feature vector. The more similar two pieces of music regarding their extracted features are, the smaller the (Euclidian) distance between their vectors is. Using those vectors the *Self-Organizing Map* (SOM) [Koh95] algorithm is applied in PlaySOM [DNR05] to organize the music files on a two-dimensional map display in such a way that similar pieces are grouped close together. The algorithm works as follows: The SOM consists of a set of units, which are arranged on a two-dimensional grid. Each of the units is assigned to a randomly initialized model vector that has the same dimension as the feature vectors. In each learning step a randomly selected feature vector is matched with the closest model vector (winner). An adaptation of the model vector is performed by moving the model vector closer to the feature vector. The neighbours of the winner are adapted as well but not as much as the model vector of the winning unit. This enables a spatial arrangement of the feature vectors such that alike vectors are mapped onto regions close to each other in the grid of the units. Once the learning phase is completed, the feature vector of each music file is mapped to its closest unit on the map.

Visualizations

In the next step different algorithms, such as e.g. the smoothed data histogram (SDH) [PRM02] algorithm, are applied to graphically visualise the SOM. Units are coloured regarding to the amount of songs per unit, starting from white (high density) over yellow to green (lower density), which can

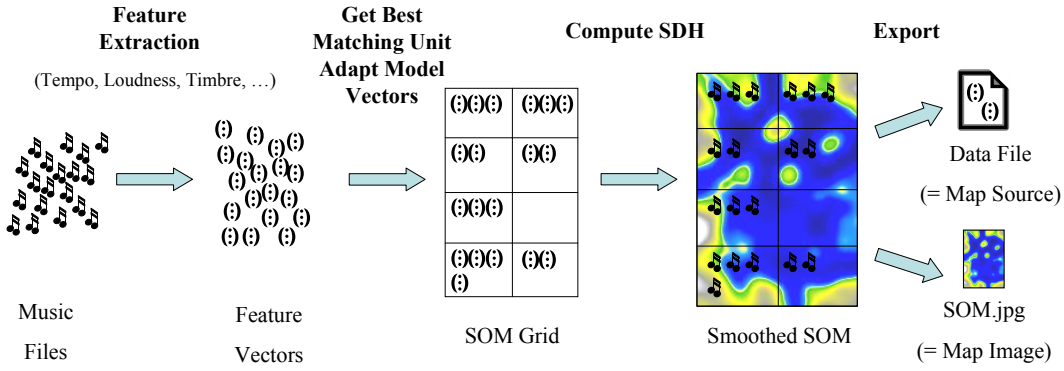

Figure 4.4: Pre-processing Content

be interpreted as mountains, where as units with fewer assigned vectors are painted blue, which is a metaphor for water (see Islands of Music [Pam01] for more details). Finally the visualisation of the SOM and the mapping between all titles with their corresponding units are exported to a jpg (=Map Image) and a data file (=Map Source), which are used as input for the MobileSOM. The SOM may consist of e.g. 280 units arranged with 20 units per row. The resolution of the jpg is 200x140 pixels. Hence every unit belongs to a square with dimensions 10x10 pixels on the map. Figure 4.4 illustrates all pre-processing stages at a glance.

4.3.6 Combined Views

The visualization techniques described above can also be combined in different layers. An example is to display a Self Organising Map with a layer containing user defined icons like shown in Figure 4.3(f).

4.4 Playlist Generation on Mobile Devices

Traditional ways of generating playlists like dragging and dropping pieces of music from a file explorer in to a player need not be suitable for mobile devices. Browsing through large directory structures on devices with small displays is a complex and inconvenient task. If a user is searching for a specific song he might want to find the piece of music by typing in the title or artist name which is, because of the lack of a keyboard on most devices, again a burdensome task. The idea of displaying a user's entire music collection as a single map on his device opens new possibilities in creating playlists more

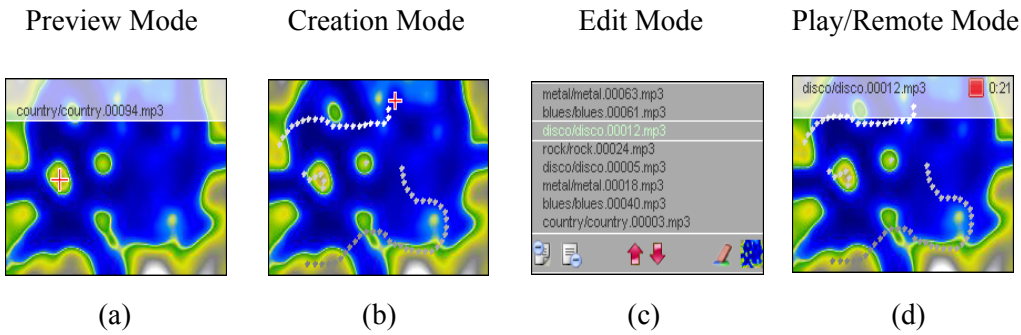

Figure 4.5: Playlist Generation

intuitively and efficiently. Like described in the previous section the map is structured that similar pieces of music are grouped together. Song titles are displayed on the screen by pointing on a specific location on the map, so the user is able to get a feeling where different styles of music are located (see *Preview Mode*, Figure 4.5(a)). After familiarizing with the map, the user can change from the preview into the playlist *Creation Mode*. By drawing a path or pointing on a single position on the map, one or more songs are selected and added to the playlist (see Figure 4.5(b)). Now just after view clicks and movements a playlist is generated quickly with the following advantages: Playlists generated in that way always contain different content, because it is hard to ever draw exactly the same path again. Users will not get bored in listening to the same songs in the same order. The playlists are more homogeneous than random generated ones because of music similarity. So the user might create different paths for different moods. Playlists are also heterogen as the trajectory covers units with different content. Accordingly the list can be altered by removing or changing the position of music titles (see *Edit Mode* in Figure 4.5(c)). The items from the final playlist can be played back from the device, streamed from a server or sent to a server (see Figure 4.5(d)). In the latter case the mobile device acts as a remote control. Figure 4.5 illustrates the stages for generating a playlist at a glance.

4.5 Summary

In this chapter the reader learned how metadata from digital audio files is generated and how this data is used to visualize music libraries on mobile devices. Songs are tagged by humans with attributes like song name, artist

or genre. This may lead to inadequate and inconsistent labelling, because people may associate different genres with the same music. To overcome this problem metadata is automatically computed from the audio signal itself. This process is called feature extraction. **Features** like timbre or rhythm are extracted and stored in a feature vector for each song. Approaches for defining a similarity measure between pieces of music are for example computing the Euclidian distance between their feature vectors or the use of collaborative filtering techniques. A music library is visualized as a music map which is split into units that contain pieces of music with similar properties (Grid-Unit Concept). A unit might be a placeholder for an album or a group of similar sounding songs. Depending on the type of content units are displayed as album covers or metaphoric symbols like emoticons that are placeholders for a specific mood. Another visualization technique is to display the grid as a topographic map where similar sounding songs are grouped together on islands. An intuitive and efficient way for generating playlists on mobile devices was described in the last section.

Chapter 5
Developing a Mobile Application

Since manufacturers of mobile devices opened their operating systems to be merged with third party software, many different platforms for developing mobile applications evolved. In the design phase of the prototype MobileSOM an important issue was to find the right setup for a suitable developing environment. The following platforms were evaluated:

.NET Compact Framework [1] is a light version of the .NET Framework that was designed to run on mobile devices such as PDAs or mobile phones. It uses class libraries from .NET Framework but also a few modules specifically designed for mobile devices. Applications are programmed in Visual Studio.NET and the resulting software runs on any device supporting the.NET Compact Framework runtime (e.g. Windows Mobile 5 devices).

Java Micro Edition [2] (formerly J2ME) technologies contain a highly optimized Java Runtime Environment that specifically addresses PDAs and mobile devices. J2ME uses a subset of classes from the Java Enterprise Edition (J2EE). Applications are programmed in freely available integrated development environments (IDEs) like NetBeans or Eclipse. The software is tested on emulators provided by the device manufacturers and runs on any mobile device that supports the K (kilobyte) Virtual Machine (KVM) which is a light version of the Java Virtual Machine (JVM).

Applications for both platforms run inside virtual machines that manage security, memory usage and runtime optimization. Both platforms come with a rich set of libraries for advanced UI (user interface), network connectivity and data management. J2ME and .Net Compact Framework allow desktop developers to migrate their skills to mobile development. For example, the Java Abstract Window Toolkit (AWT) UI can be directly used in certain J2ME profiles, and the Windows Forms controls can be used in Compact Framework applications. Many generic IDEs, development toolkits, and command-line tools are available from leading vendors in the J2ME

[1] http://msdn2.microsoft.com/en-us/netframework/aa497273.aspx
[2] http://java.sun.com/javame/index.jsp

space. Microsoft's flagship development tool, Visual Studio .Net, is also fully integrated with Compact Framework.

The reasons for using J2ME as programming language were on one hand the existing programming skills by the author and on the other hand the simplicity of setting up a complete developing environment. Moreover J2ME runs on a range of devices from Smartphones to Set-top boxes from all major vendors, while .Net Compact Framework runs only on devices from Microsoft Pocket PC licensees. This chapter gives an overview about the J2ME technology, describes the features of the used IDE NetBeans[3] and discusses different emulators by various manufacturers. In the end a tutorial of how to create a "Hello World"-MIDlet is demonstrated.

5.1 J2ME

Java Platform, Micro Edition or Java ME (formerly referred to as Java 2 Platform, Micro Edition or J2ME), is a collection of Java APIs for developing applications for resource-constrained devices such as PDAs or mobile phones. Java ME is formally a specification, although the term is frequently used to also refer to the runtime implementations of the specification. Java ME was designed by Sun Microsystems and is a replacement for a similar technology, PersonalJava[4]. Sun Microsystems has tended not to provide free binary implementations of its Java ME runtime environment for mobile devices, rather relying on third parties to provide their own, in stark contrast to the numerous binary implementations it provides for the full Java platform standard on server and workstation machines. J2ME architecture is designed to be modular and scalable. This modularity and scalability are defined by J2ME as three layers of software built upon the Host Operating System of the device:

Java Virtual Machine This layer is an implementation of a Java virtual machine that is customized for a particular device's host operating system and supports a particular J2ME configuration.

Configuration The configuration is less visible to users, but is very important to profile implementers. It defines the minimum set of Java virtual machine features and Java class libraries available on a particular group of devices. In a way, a configuration defines the "lowest common denominator" of the Java platform features and libraries that the developers can assume to be available on all devices.

[3] http://www.netbeans.org/

[4] http://java.sun.com/products/personaljava/

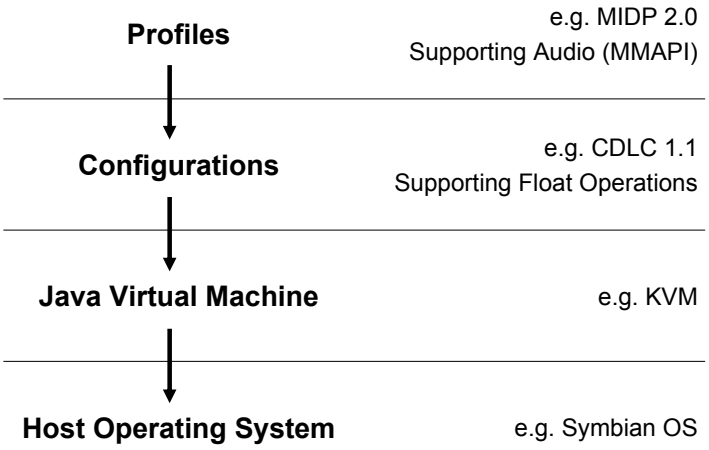

Figure 5.1: J2ME software layer stack

Profile The profile is the most visible layer to users and application providers. It defines the minimum set of Application Programming Interfaces (APIs) available on a particular "family" of devices. Profiles are implemented upon a particular configuration. Applications are written **for** a particular profile and are thus portable to any device that **supports** that profile. A device can support multiple profiles.

Figure 5.1 shows the layer architecture of J2ME at a glance. The Java Virtual machine running on the host operating system of a mobile device is implemented by the device manufacturers themselves. A Configuration defined by the Java Community Process ensures that the implementation of the JVM supports a certain set of classes and libraries. On top of a specific Configuration sits a Profile which implements the API's that are visible to the programmers.

5.1.1 Connected Limited Device Configuration

The Connected Limited Device Configuration (CLDC) is a specification of a framework for Java ME applications targeted at devices with very limited resources like PDAs and mobile phones. Today there are two configurations for mobile devices: The CLDC 1.0 which was developed under the Java Community Process as JSR 30 and the CLDC 1.1 developed under JSR 139. The CLDC 1.0 did not support any form of floating point arithmetic, which results in a lack of mathematical functions like *sqrt*, *sin*, *cos*.This

limitation was fixed with the CLDC 1.1. Requirements for devices using this configuration are a 16-bit CPU, a total of 160 KB memory available to the JVM and a limited connection to some kind of network.

5.1.2 Mobile Information Device Profile

The mobile Information Device Profile (MIDP) is a specification published for the use of Java on embedded devices such as mobile phones and PDAs. MIDP is part of the Java ME framework and sits on top of a "configuration", such as the Connected Limited Device Configuration. It was developed under the Java Community Process as JSR 37 (MIDP 1.0) and JSR 118 (MIDP 2.0). As of 2006, MIDP 3.0[5] is being developed under JSR 271. Any devices only supporting MIDP 1.0 are not sufficient for rich multimedia applications and therefore cannot be used for MobileSOM. In MIDP 2.0 this limitations were solved by adding the following API's:

javax.microedition.media The Multimedia API (MMAPI) specified in JSR 135 contains the base classes of the multimedia playback.

javax.microedition.io.file The FileConnection API specified in JSR 75 gives access to the local file systems on a device.

For some devices there also exist vendor-specific APIs like the "Nokia User Interface 1.0". They are not part of a profile and using them obviously reduces the portability of an application and is not recommended if the application is targeting the horizontal market of mobile devices.

5.1.3 A MIDlet

A MIDlet is a Java program for embedded devices that is executed by the device's Java Virtual Machine. In general these are commercial games and applications but also prototypes for realising innovative concepts like MobileSOM. A MIDlet requires a device that implements at least one of the standardized MIDPs in order to run. Like other Java programs, MIDlets in theory have a "compile once, run anywhere"-potential. This is not always the case, because devices have specific firmware bugs or incomplete API implementations. To write a MIDlet, the Sun Java Wireless Toolkit from the Java website[6] can be used. It is available on several platforms and is completely free. MIDlet distributions also consist of a .jad file describing the contents of the JAR file.

[5] http://jcp.org/en/jsr/detail?id=271
[6] http://java.sun.com/products/sjwtoolkit/download.html

A MIDlet has to fulfill the following requirements in order to run on a mobile phone:

- The main class needs to be a subclass of javax.microedition.midlet.MIDlet
- The MIDlet needs to be packed inside a .jar file
- In some cases, the .jar file needs to be signed to access specific system resources.

A .jar file may contain more MIDlets. A pool of MIDlets is called a **MIDlet Suite**.

5.1.4 Signing a MIDlet

In case a MIDlet wants to access specific system resources like the file system, the user is explicitly asked to allow this by the operating system. To avoid confirming manually those message boxes all the time, a MIDlet is signed. The procedure for signing MIDlet suites works like this: First a key pair (containing a public and a private key) is generated in the J2ME Wireless Toolkit in the MIDlet signing window. Then a Certificate Signing Request (CSR) is generated which is sent to an official certificate authority (CA) like Verisign [7]. The CA needs personal information to verify someone's identity. After paying a fee a certificate from the CA that certifies the public key is sent to the programmer. Now this certificate is imported into the J2ME Wireless Toolkit and the programmer can use his own private key to sign MIDlet suites. The J2ME Wireless Toolkit will take care of the details of placing the certificate into the MIDlet suite. Detailed Information can be found in the Java Wirelss Toolkit User's Guide in Chapter Security and MIDlet Signing[8].

5.1.5 MMAPI

The Multimedia Java API (MMAPI) is an API specification for the Java ME platform targeting mobile phones. Depending on how it is implemented, the APIs allow applications to play or record sounds and video from different file formats. MMAPI was developed under the Java Community Process as JSR 135.

[7] http://www.verisign.com/products-services/security-services/code-signing/index.html

[8] http://java.sun.com/j2me/docs/wtk2.2/docs/UserGuide-html/security.html

Implementation

As with most Java ME specifications, implementations differ despite the best efforts of the specification authors to ensure consistency. An obvious area for differences are the accepted file types. In Table 5.1 on page 48 a list of Nokia devices with their playable audio types is shown[Nok07]. Even-though the devices are from the same brand, there is a huge diversity in the MMAPI implementations. More obscure areas are whether mixing is supported (e.g. playing a MIDI track and layer PCM sound effects on top). Another source of extreme variance is in performance. For example, if an HTTP clip is requested, at what point does the clip get downloaded? The specification recognises this by providing two Player methods that can be called in advance of actually playing: realize and prefetch. Depending on the implementation, these may do some of the work of getting the clip into a playable state, thus making it quicker to actually play the clip when it is needed. Some implementations are sophisticated enough to actually stream a clip on request whilst it is being played. Symbian OS contains a very complete implementation of JSR 135, but even this is highly dependent on the underlying multimedia capabilities of the device, and some device manufacturers may choose not to expose the more obscure parts of Java ME such as recording.

5.1.6 User-Interface API

Programs written in J2ME are distinguished between applications running in the background or the ones that interact with a user. To design User Interfaces (UI) for MIDlets the package "javax.microedition.lcdui" is used. The developer can choose between a Low- and a High-Level-API (see Figure 5.2).

High-Level-API

The High-Level-API consists of form elements like text fields, lists and alert boxes. Those elements cannot be customized and there are only rudimentary layout functions for placing the elements on the screen available.

Low-Level-API

With the Low-Level-API pixels are addressed directly. Programming user interfaces is more sophisticated but the results are more pleasing like the comparison of two user interfaces in Figure 5.3 shows. All painting opera-

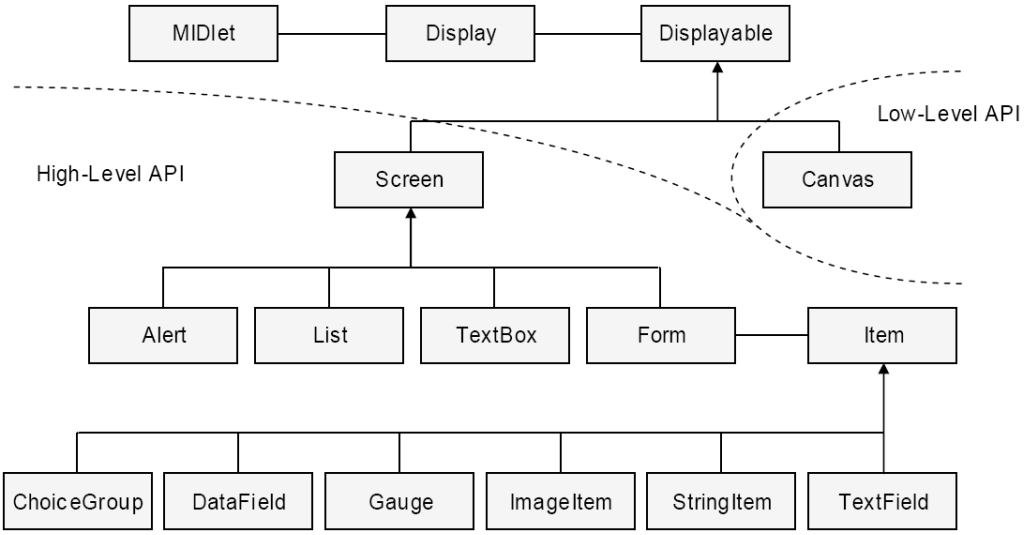

Figure 5.2: Low- and High-Level-API of the MIDP

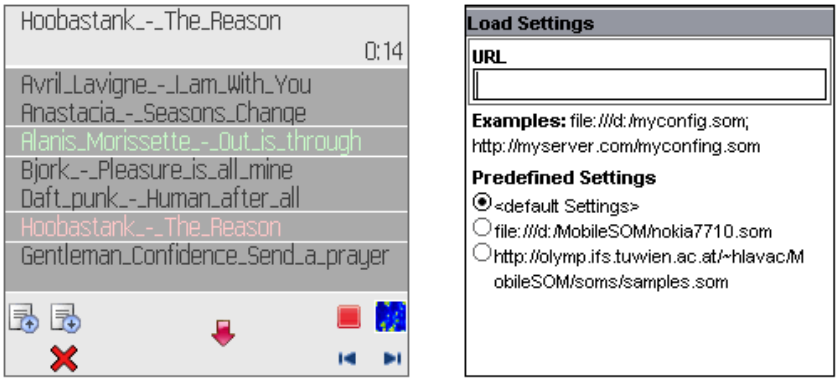

Figure 5.3: Comparison of a User Interface designed with the Low-Level-API (left) and an interface composed with the High-Level-API (right)

tions are available in the "Graphics"-Object which is rendered in the "paint"-Method of the "Canvas"-Object. Those operations include drawing pixels or complex objects like circles, filled bars and images. Images in general are stored as separate files in the .jar file of the MIDlet. It is important to keep their file size as small as possible because of heap limitations. The native format used for images is .png but on most devices .jpg decoding is possible too.

Both APIs inherit from the "Displayable"-Object which is set in the "Display"-Object with the "setCurrent()"-Method. The "Displayable"-Object is the parent class of the "Screen"-Object (High-Level-API) and the "Canvas"-Object (Low-Level-API). The following code snippet uses the High-Level-API element "alert" which shows a message on the display of a device.

```
1 import javax.microedition.lcdui.*;
2 ...
3 Alert alert = new Alert('myAlert');
4 alert.setString('Hello!');
5
6 Display display = Display.getDisplay(this);
7 display.setCurrent(alert);
```

5.1.7 Record Management System (RMS)

The Record Management System (RMS) is used to permanently store data on a device. It allows an application to save and retrieve data from the device without the need for accessing the file system. The same technique is used by a browser saving information (cookies) from a user session on the local machine. The web server can read this information anytime the user visits the website again. The storage of custom data on a device is persistent. Even if the battery is changed the data will still remain on the device.

5.2 IDE

For developing a mobile application using an Integrated Development Environment (IDE) is inescapable. Various free available IDEs for creating software written in J2ME like NetBeans[9] or Eclipse [10] exist. Prerequisites for using a specific IDE are seamless integration of external emulators for

[9] http://www.netbeans.org/
[10] http://www.eclipse.org/

testing purposes, configuration options in the source code and deploying facilities to real world devices. The IDE used by the author for developing MobileSOM was NetBeans and its features are described in the following sections in more detail.

5.2.1 NetBeans

NetBeans is a fast and feature-rich tool for developing Java software. It is standards-compliant and runs on any operating system where a Java Virtual Machine is available. While the IDE is modular (its functionality can be extended by plug-ins), the focus is on providing all the tools a Java developer needs in one download, with no additional set-up or configuration needed to begin productive work quickly. The base product includes support for desktop (AWT/Swing), web tier (Servlets/JSP/JSF/Struts) and Java Enterprise Edition (EJB and web Services) development, and bundles a database and Java EE application server. The NetBeans IDE is open-source and can be used free of charge for any type of software development.

5.2.2 NetBeans Mobility Pack

The NetBeans Mobility Packs add to the NetBeans IDE everything needed to immediately start writing, testing and debugging Java applications for mobile phones and other Java Micro Edition (Java ME) technology-enabled devices. The NetBeans Mobility Pack provides comprehensive support for the Connected, Limited Device Configuration (CLDC) 1.1, Mobile Information Device Profile (MIDP) 1.0 and 2.0, and includes visual design tools for mobile applications, integrating mobile applications with web services. It includes the Java ME wireless toolkit and device emulators, so that no additional downloads are needed to start working with mobile technologies even though it is easy to integrate third-party emulators [11] and SDKs for a robust testing environment.

5.3 Emulators

Emulators are used to test a MIDlet Suite on the PC first before it is deployed on a real world device. Besides the Java Wireless Toolkit third-party emulators from various manufacturers like Nokia or Sony Ericsson, which have extended features like on-device-debugging, exist. The following section describes the emulators that were used to test MobileSOM in more detail.

[11] http://www.netbeans.org/kb/50/midpemulators.html

5.3.1 Java Wireless Toolkit

The Sun Java Wireless Toolkit [12] (formerly known as Java 2 Platform, Micro Edition (J2ME) Wireless Toolkit) is a state-of-the-art toolbox for developing wireless applications that are based on J2ME's Connected Limited Device Configuration (CLDC) and Mobile Information Device Profile (MIDP), and designed to run on cell phones, mainstream personal digital assistants, and other small mobile devices. The toolkit includes the emulation environments, performance optimization and tuning features, documentation, and examples that developers need to bring efficient and successful wireless applications to market quickly.

5.3.2 Carbide.j

Carbide.j (formerly Nokia Developer's Suite for J2ME) [13] is a software development tool for Java Platform, Micro Edition (Java ME) developers that enhances the development and verification of applications for Nokia devices. It provides tools for creating Mobile Information Device Profile (MIDP) and Personal Profile (PP) applications and deployment packages, signing applications, and deploying applications to devices. It is also an essential tool for managing, configuring, and running emulators for various Nokia platform and device SDKs.

SDK vs. Prototype SDK

SDKs emulate the phone behaviour of Nokia devices as closely as possible, because their emulators are based on the same software as the specified Nokia devices.

The emulators of a prototype SDK are based on the reference implementation of the Java APIs of the corresponding Platform. The main benefits of prototype SDKs are early availability and fast performance. Each prototype SDK package contains several Platform emulators to create and compile an application and verify its functionality for different Platforms and devices.

5.3.3 Sony Ericsson SDK

This suite [14] of Java ME tools supports Java MIDP 1.0, MIDP 2.0, Java 3D API and Javadoc for the Sony Ericsson handsets. The SDK supports Java

[12] http://java.sun.com/products/sjwtoolkit/
[13] http://www.forum.nokia.com/main/0,6566,1_84,00.html
[14] http://developer.sonyericsson.com/site/global/docstools/java/p_java.jsp

Micro3D emulation and includes all the necessary tools to support On Device Debugging for mobile applications.

5.4 A "Hello World"-MIDlet

This tutorial shows the reader how to create and run a "Hello World"-MIDlet within the NetBeans IDE. In a later step the functionality of the MIDlet is extended by adding a music player that is able to stream a wav file from any given URL. Both, the NetBeans Mobility Pack and NetBeans IDE, have to be installed first. After starting the IDE the "New project" item is chosen from the file menu like shown in Figure 5.4. A project wizard opens up and from the category "Mobile" the Project "Mobile Application" is chosen (see Figure 5.5). The "Next"-Button is pressed and on the following page the name and the location of the project are set. If the "Finish"-button is clicked the wizard creates a dummy mobile application with the default java wireless toolkit simulator selected. A click on the "Play"-button (see Figure 5.6) packs the application in a .jar file and runs the MIDlet in the emulator. The program displays a "Hello World" message on the screen. Now the MIDlet is extended with a simple audio player. In the project explorer, the package "hello" is expanded and the "HelloMidlet" class is double clicked like shown in Figure 5.7. In the source code window the MMAPI is imported by adding the line

```
1 import javax.microedition.media.*;
```

to the other import clauses on top of the code window. The method "initialize()" is complemented with the following code for playing an audio file.

```
2         Player p;
3         try {
4             p = Manager.createPlayer(
5             "http://www.ifs.tuwien.ac.at/mir/" +
6             "pocketsom/mobilesom/wav/pop.wav");
7             p.start();
8         } catch (MediaException ex) {
9             ex.printStackTrace();
10        } catch (IOException ex) {
11            ex.printStackTrace();
12        }
```

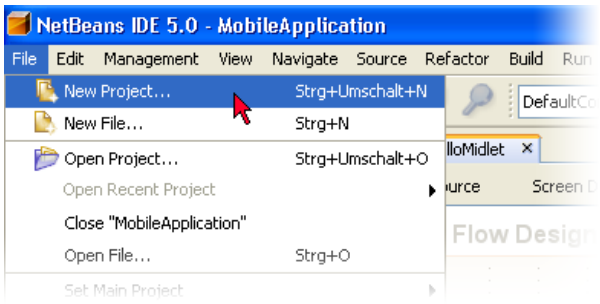

Figure 5.4: Creating a new mobile project in NetBeans

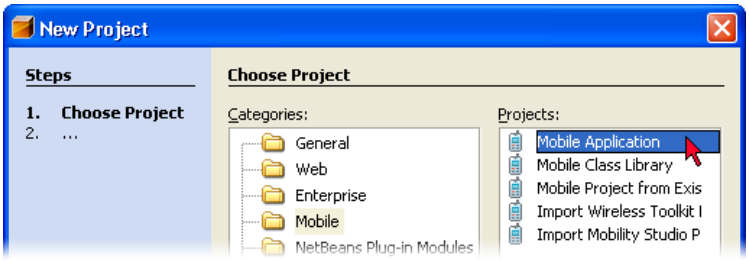

Figure 5.5: NetBeans Project Wizard

After starting the application the MIDlet wants to access the network, which has to be confirmed by the user one time. If the network resource exists the file is downloaded to the device and played back.

5.5 Summary

In this chapter the reader learned about the software architecture of J2ME. The most important APIs for creating rich multimedia applications like the Multimedia API or the Low-Level User Interface API were introduced. A mobile application in J2ME is called a **MIDlet**. The IDE NetBeans assists the programmer in developing such MIDlets, testing them on third-party emulators and deploying them to real world devices. A brief tutorial at the

Figure 5.6: Start the MIDlet in the emulator

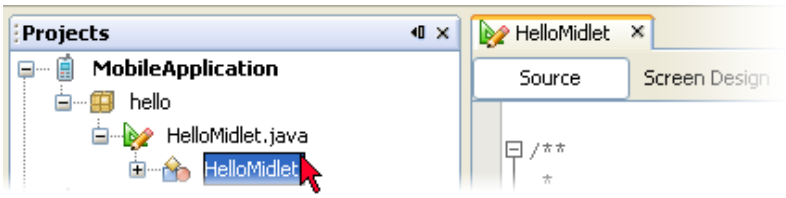

Figure 5.7: Edit Source Code of the MIDlet

end of this section demonstrated how to create a “Hello World”-MIDlet and how to deploy and run it in the Java Wireless Toolkit emulator.

Platform baseline	**Real Audio**	**MP3**	AAC	WAVE	AU
devices Extensions	.ra,.rm	.mp3	.aac, .mp4 .3gp, m4a	.wav	.au
Series 40 2nd Edition:					
3220, 5140i, 6230	-	-	-	-	-
6230i	-	x	aac	1)	-
Series 40 3rd Edition:	-	x	aac	1)	-
6111	-		aac		-
6280	-	5)	also m4a	1), 2)	-
7370	-	5)	also m4a	1), 2)	-
S60 1st Edition:	-	-	-	2)	-
3650, N-Gage QD	-	-	-	2)	-
S60 2nd Edition:	x	x	aac	x	x
3230					
6600		-	-	2),3)	3),4)
7610			also 3gp		
6630, 6680			also 3gp, mp4		
N70, N90			also 3gp, mp4		
S60 3rd Edition:					
E60	x	x	aac, 3gp, mp4	x	x
Series 80 2nd Edition:					
9300, 9500	x	x	aac, 3gp	x	x
Nokia 7710:	x	x	aac	x	x

Table 5.1: Audio capabilities of Nokia Device: x = Tested to work. 1) Tested OK using the audio/wav MIME type. 2) Tested OK using the audio/x-wav MIME type. 3) Tested OK using the audio/basic MIME type. 4) Tested OK using the audio/au MIME type. 5) The device should also support ID3v2 metadata for MP3 and AAC (three tags supported: title, artist, URL). MIDP: Mobile Media API Support In Noki a Devices 3 (11)

Chapter 6
MobileSOM

MobileSOM[1] is a prototype entirely programmed in Java Micro Edition (J2ME). It allows users to browse through their music collection that is displayed as a two-dimensional map on their mobile device. Pieces of music are played locally or are streamed from a web server. Moreover, the prototype can be used as a remote control. Instead of playing a piece of music on the mobile device, the software triggers another device to play the song.

MobileSOM offers a framework for using different visualisation styles of music libraries based on the Grid-Unit Concept introduced in Section 4.2 on page 27. It is the first J2ME application available that implements the paradigm of generating playlists using a self organising map.

MobileSOM is not designed for finding a specific piece of music, as there are no special search mechanisms implemented for locating exactly the specific song a user has in mind. MobileSOM assists in generating playlists that fit into a specific mood or style. The software supports the user exploring music content rather than looking for the latest Britney Spears hit.

MobileSOM was built from scratch. No existing frameworks like J2ME Polish[2] or other software libraries were used. This chapter contains detailed description of the software architecture, followed by an installation guide and a section on how to set up a test environment for deploying and running the software on real world devices. The reader will learn about the user interface and how to interact with the software. At the end of the chapter practical experiences of running the software on real world devices and ideas for extending the software with new features are given.

6.1 Software architecture

The software MobileSOM is grouped into packages and resources. All files are packed into a .jar file, called "MobileSOM.jar", which will be installed on a mobile device in a later step. The structure of the .jar file is separated in

[1] http://www.ifs.tuwien.ac.at/mir/pocketsom/mobilesom/
[2] http://www.j2mepolish.org/

- Resource folders containing data files
- Package folders containing classes in context of their package name
- a META-INF directory containing the manifest file

In the following section an overview of the classes and resources used is given. Detailed information about all classes can be found in the Java Documentation of MobileSOM[3].

6.1.1 Resources

Resources are files with either binary (e.g. images) or textual data (e.g. configuration files), that are packed together with the classes into the jar file and loaded into memory if needed during runtime. They can be exchanged or modified before the software is installed on the mobile device. MobileSOM uses the following resources:

- Icons
- a map representing the underlying items (=Map Image)
- a data file mapping each item to a location on the map (=Map Source)
- a configuration file
- a dummy mp3
- and a language file (by default English)

Icons are symbolic menu items, that are displayed dynamically on the screen, depending which user actions are possible at a certain state. Figure 6.1 shows all icons, stored as gif files in the "\img" folder of the .jar file. The default Map Image (see Figure 6.2) is the visual representation of the data stored in the Map Source and is encoded as a jpeg file. It is loaded when the software is started for the first time and then transferred to the RMS of the mobile device. The Map Source is a text file using the following syntax:

```
1 5 // Number of Columns in the Grid
2 4 // Number of Rows in the Grid
3 -
4 0 0 Hoobastank - The Reason.mp3
```

[3] http://www.ifs.tuwien.ac.at/mir/pocketsom/mobilesom/doc/

```
0 0 Lenny Kravitz - Fly away.mp3
    . . ...
```

The first three lines specify the layout of the Grid, which divides the Map Image into Units, where as the rest of the file contains the Map Items (one per line). The first two values assign the Map Item to a Unit followed by the name of the item. Once the file is loaded its items and their locations are stored to the RMS as well. The configuration file "conf.properties", stored in the "\resources" folder, contains the initial values of the following properties:

- the path to the Map Source
- the path to the Map Image
- the mp3 server
- the remote server (if device is used as a remote control)
- the path to the local mp3 directory (on device or memory card)
- the mode (whether the application is used in demo(0), live(1) or remote(2) mode)
- the language
- the boolean value "randomize unit items" (if set to 1 items from a Unit are returned randomly, rather than in the sequential order in which they are listed)

On the basis of the following configuration example its properties are described in more detail.

```
mapimage=/resources/demo.jpg
mapsource=/resources/demo.list
mp3server=http://mymp3server.com/
remoteserver=http://remote.com/makeplaylist.php?song=
localmp3directory=file:///d:/MobileSOM/mp3/
mode=1
playlistname=standard
language=en
randomizeunititems=0
```

The default “mapimage” and “mapsource” are loaded from the resources folder within the jar file. The URL of the “mp3server” points to a location in the web from where the music files, which are defined in the Map Source, are streamed. The address of the “remoteserver” takes as an argument the song to be played on the remote machine (e.g. ?song=kelis-milkshake.mp3). The “localmp3directory” points to a location on the mobile device where the audio files are stored. MobileSOM tries to read the files from the local directory first. If they are not available on the device, the software connects to the Internet to stream the songs from the mp3 server. In case a piece of music is neither on the device nor in the web present a “Song not found. Check your Settings!” message is displayed to the user. If the property “mode” is set to 1, MobileSOM is playing songs on the device. In mode 2 the songs titles are sent to the remote server and played on another device. If the property is set to demonstration mode (mode=0) no songs are played at all. This mode is used to test the user interface. The “playlistname” is used to identify a user’s playlist on the remote server. If multiple users are streaming from the same remote server their playlists will not get mixed up. The “language” property is set to English (“en”) by default, which assumes a language file “en.properties” to be present in the “\resources” folder. The parameter can be set to any other language (“xx”) as long the corresponding “xx.properties” file exists in the .jar file. The last configuration parameter “randomizeunititems” determines that songs picked from a unit are either played back sequential (0) or by random (1).

Another resource is the dummy mp3 which is a very short silent audio file used for pre-buffering. On some devices playing a long mp3 file takes the audio player about one minute to initialize the song before it can be played. In case the short dummy file is played before, the initialization time is reduced to about one second. Last but not least there is the language resource file. It contains all text elements displayed to the user during runtime. The default language used is English. An advantage of having all text elements in a separate file is that no changes in the source code have to be done when switching to a different language.

The following subsections give a detailed description of all classes used by MobileSOM

6.1.2 Package IO

This package includes classes for accessing the local files system, the Record Management System (RMS) and the Internet via the HTTP protocol.

Figure 6.1: Menu Icons

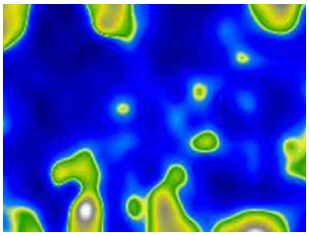

Figure 6.2: Default Map

Class ResourceHandler

The Resource Handler manages to fetch resources from the jar file itself or from an external location like the local file system, the web or the RMS.

Class FileHandler

This class is used for dealing with file operations. Files can be loaded into memory as a String or a DataInputStream. Depending on the mobile device or emulator used files are accessed in different ways. To read files from Sony Emulators the URL

```
1 file:///root1/filename.ext
```

is used. Files on a Nokia 7710 are accessed with

```
1 file:///c:/filename.ext
```

Class HttpHandler

This class handles HTTP traffic between the mobile device and any server in the web containing music content or data files. Requesting response from an URL is a blocking process. As long the mobile device is waiting for response no user interaction (e.g. clicking on a button) can be done. For

that reason the class is implemented as a thread running in the background so the program flow will not be interrupted. The class can be used in one of the two ways:

- calling an URL asynchronously not waiting for any response. Such calls are done when using the device as a remote control (e.g. sending a STOP command to the remote machine to stop playing a piece of music)
- using the static method "getResponseFromURL(url)" to get the response as a byte stream or converted to a string from a given URL. The method is called when requesting configuration or data files from the web (e.g. the Map Source or the Map Image).

Class RMSHandler

The class RMSHandler is used to read and store variables permanently on the mobile device using the Record Management System (RMS). There are two record stores saved on the device. One is used to save the properties (settings) of the Application as key value pairs in the property object. (e.g. key: "mapimage", value: "gztan.png"). The other one saves binary data like the mapping of items to units (Map Source) or the graphical representation of the items (Map Image).

6.1.3 Package Manager

This package manages the run-time configuration of the software in setting up the language and the properties stored on the device.

Class LanguageManager

This class enables multi language support. The language is set in the properties manager. Labels in the program code are not hard coded but they are read from the "language.properties" file (e.g.:en.properties). A language file looks like this

```
1 ...
2 #Alert Labels
3 ALERT_SUCCESS=Success
4 ALERT_ERROR=Error
5 #Messages
6 MESSAGE_SETTINGS_LOADED=Settings are loaded successfully
```

```
7 ...
```

Those labels are used in the source code like in the following example

```
1 text.setText(languageManager.MESSAGE_SETTINGS_LOADED);
```

Class PropertyManager

This class is used to read, write and initialize properties for running the software. If the application is executed on the mobile device for the first time, the default properties (declared in the resource "conf.properties") are loaded and then stored to the RMS. If settings are changed by the user at runtime the properties in the RMS are overwritten with the new values. From now on every time the software is started properties are read from the RMS, like a cookie from a web browser when visiting a specific page.

Class Properties

The Properties class holds all properties as key-value pairs. They can be accessed with "getProperty(String key)" or "setProperty(String key, String value)".

6.1.4 Package Map

This package is a collection of classes that implement the Grid-Unit concept introduced in chapter 4.

Class Unit

A Unit is a collection of Items that have similar properties. For example a unit contains pieces of music that sound similar or pieces of music from the same artist. Every unit can be labelled to describe its content.

Class MapItem

A Map Item is an item that belongs to a Unit on the Grid, saved with its name (e.g. the filename of an mp3 file). Figure 6.3 shows Unit(2,1) with x Map Items.

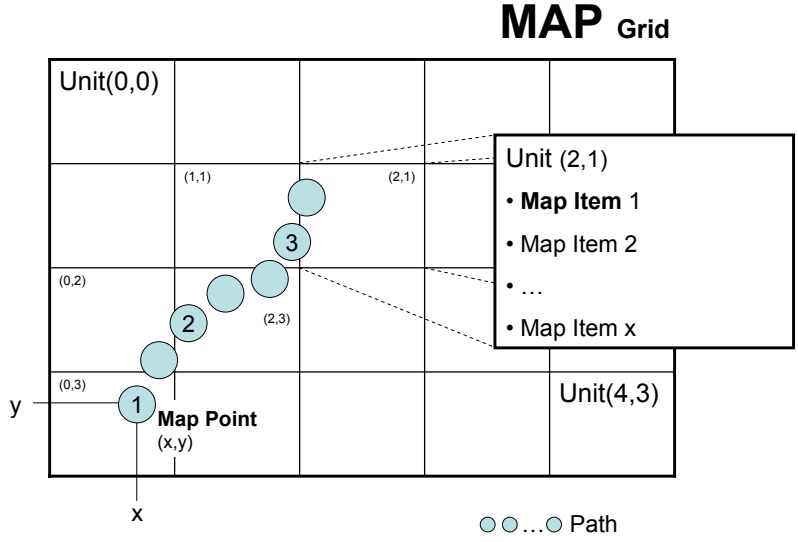

Figure 6.3: A MAP Grid with 7 Map Points. Map Point 1 has the coordinates (x,y) and is mapped to Unit (0,3). Unit(2,1) has x Map Items.

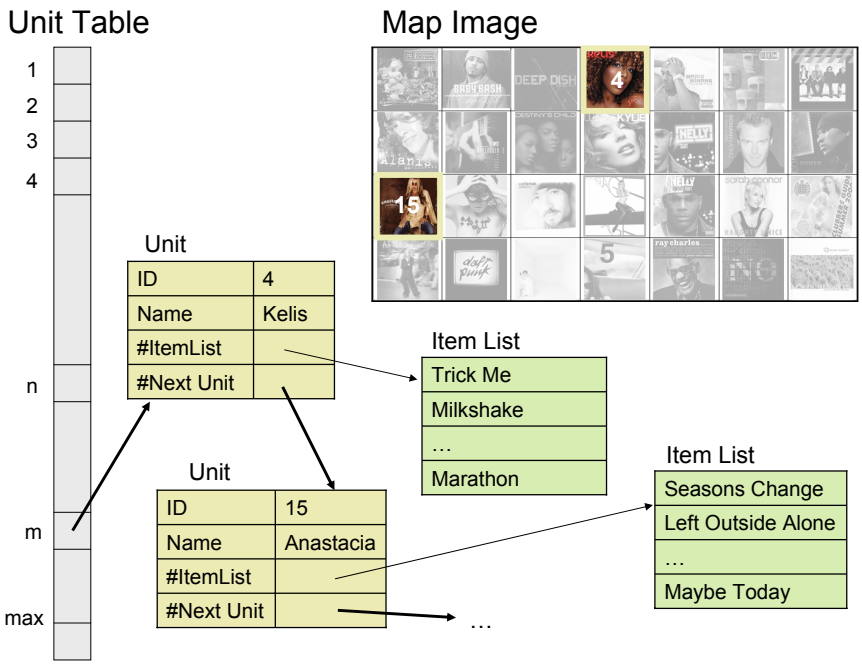

Figure 6.4: The Unit Table implemented as a Hash Table and the corresponding Map Image with the Units highlighted shown in the Hash Table

Class Grid

A Grid is divided into a set of Units (Figure 6.3). The first Unit is in the upper left corner with the values (0,0). Every Unit in the grid is assigned to a Unit Id that is put together by multiplying the row of the Unit with the total number of Units per row and adding the column of the Unit + 1. A Unit contains one or more Map Items but can be empty as well. Map Items with their corresponding Unit are loaded the first time from the default Map Source and then from the RMS directly into the Unit Table (Figure 6.4). The Unit Table is implemented as a Hash Table with the Unit ID as the key and the Unit Object itself as the value. Every Unit has a Vector holding the Map Items.

Class MapPoint

A Map Point is a point on the Grid that is assigned to a specific Unit, created for example by tapping the stylus on a specific location on the Grid. Coherent Map Points (see Figure 6.3) are stored in a vector (= Path). The path is used for creating MapItemLists in later step. **Note:** A Map Point is not a

Map Item!

Class MapItemList

The class MapItemList extends a vector holding a collection of Map Items. A Map Itemlist is generated by assigning each Map Point to a Unit from the Grid. In Figure 6.3 Map Point 1 is mapped to Unit (0,3). From that Unit a predefined collection of Map Items is taken and added to the Map Itemlist. The procedure is repeated through all Map Points (=Path). The algorithm for building a Map Item List is printed in Appendix A.1 and works as follows: The method "build" is called to build a Map Itemlist from the path that was drawn by the user. In an iteration every single Map Point is fetched with its corresponding Unit. From that Unit a vector of Map Items is retrieved. This can be an empty vector or a vector with one or more Map Items. If the number of Map Items is smaller than the maximal allowed number of Map Items per Unit (parameter "maxNumberOfMapItems") all Map Items from the vector are added to the Map Itemlist. Otherwise the maximal allowed number of Map Items are added. The Map Itemlist may not contain any Map Item twice, i.e. a Map Item that is already in the Map Itemlist and that shows up again in any other Vector, will not be added. Map Items are chosen from the Vector either randomly if the parameter "bRandomize" is set to true or sequentially. The default value for "maxNumberOfMapItems" is 1, i.e. that each Map Point corresponds to maximal one Map Item.

6.1.5 Package Player

This package contains a class Audioplayer that is able to stream or play items from the device itself or from the network.

Class AudioPlayer

The class is implemented as a thread to sustain user interaction during playback. The player can play a single Map Item or a list of items (= MapItemList). Depending on the configuration the player streams audio files from a server or plays them from the device. The player can also act as a remote control, i.e. instead of playing a piece of music, the player will trigger another device to play the song. Possible audio formats are .wav or .mp3 depending on the implementation of the MMAPI for a specific device.

6.1.6 Package Utils

This package contains utilities for Graphic and String manipulation.

Class StringUtils

This class provides methods for manipulating strings, like a "replace"-function, that is not supported by the J2ME framework by default. The method "URLEncode" encodes a given string to an URL. Alphanumeric letters [0-9a-zA-Z], special characters $-_.+!*'(), and reserved characters used for their reserved purposes are left unencoded within a URL. All other characters are encoded as Hex Values.

Class DrawUtils

The class DrawUtils is a helper class for showing alerts and rescaling or blending images.

6.1.7 Package MIDlet

This package contains all MIDlets that are used for user interaction in MobileSOM.

Class ChangeSettings

This MIDlet lets the user modify the current settings that were described in section 6.1.1.

Class LoadSettings

Settings are stored in configuration files with the extension .som. This MIDlet gives the user the opportunity to load such files from a customised URL (file:/// or http:///) or to choose between different predefined Settings. Figure 6.5 shows two screenshots of the Settings Dialog "ChangeSettings" and "LoadSettings". Both MIDlets use the High-Level API of J2ME.

Class MapViewer

The MapViewer is the main MIDlet of the application MobileSOM in which the user can interact with the Map Image and the music items that are placed on the map. The display variable of the MIDlet is set to an instance of the MapCanvas class.

Class MapCanvas

The MapCanvas extends the Canvas class which is the entry point for a user interface using the Low-Level API. MapCanvas displays the Map Image or

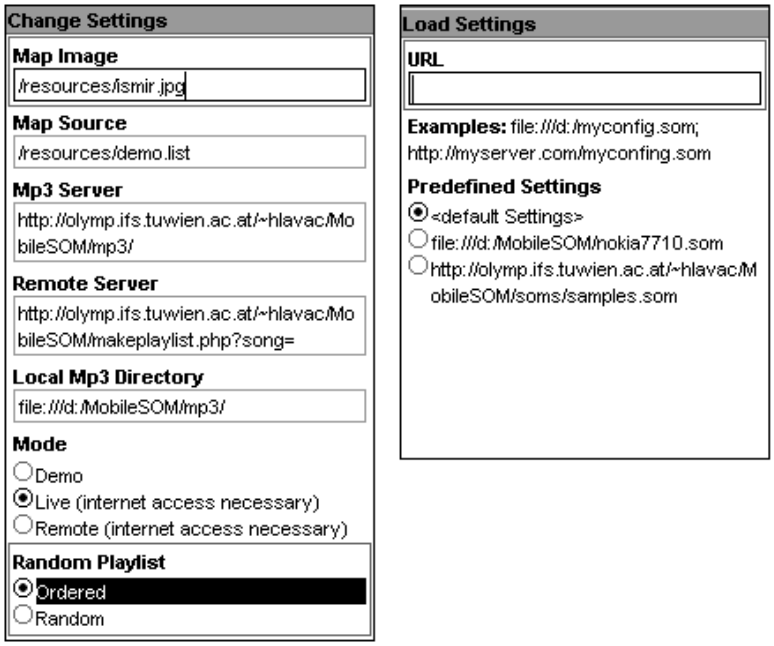

Figure 6.5: The High-Level API user interfaces to load new or modify current settings.

the Playlist Editor. It handles pointer events (e.g. clicking a menu item or drawing a path) and key strokes.

6.1.8 Manifest

The manifest of a MIDlet Suite gives the mobile device information about the minimal configuration needed in order to run the containing MIDlets. It also includes the vendor name, the version, the MIDlet labels and the name of the suite itself. The manifest file of MobileSOM is shown in the following listing:

```
1 Manifest-Version: 1.0
2 ...
3 MIDlet-3: Load Customized Settings,,midlet.LoadSettings
4 MIDlet-2: Change Settings,,midlet.ChangeSettings
5 MIDlet-1: Map Viewer,,midlet.MapViewer
6 MIDlet-Vendor: IFS - TU Wien
7 MIDlet-Version: 2.0
8 MIDlet-Name: MobileSOM
9 MicroEdition-Configuration: CLDC-1.1
```

```
10 MicroEdition-Profile: MIDP-2.0
```

There are three MIDlets “Load Customized Settings”, “Change Settings”, “Map Viewer” included. The minimal device configuration needed is the CLDC-1.1 and the required device profile is the MIDP-2.0.

Figure 6.6: XPlayer: The mobile device sends a remote call to play a song which is processed by the XPlayer

6.2 XPlayer: MobileSOM as Remote Control

MobileSOM can be used to make a mobile device work like a remote control. The prototype sends remote calls (e.g. for playing a music file) which are processed by the XPlayer. The scenario is shown in Figure 6.6. XPlayer is realized as a web page[4] containing an applet that communicates with the remote server and an applet that plays the audio files. In case a user selects a song on his mobile, the URL of the song is sent to the remote server. The server creates a playlist which is a text file containing the URL of the song. The playlist is checked by the communication applet every two seconds. In case there is a new entry, the audio player applet streams the song from the given URL. The playback is stopped by either clicking the stop button on the mobile device or in the web browser. For security reasons the location of the applet has to be on the same server where the mp3's are streamed from. Otherwise a cross scripting error prevents the applet from accessing files on a different machine. Java does not support native playback of mp3's. The applet includes a library from the "javazoom"-project[5] to decode audio files in a PCM stream that is then processed by the Java sound system.

[4]http://www.ifs.tuwien.ac.at/mir/pocketsom/mobilesom/xplayer/XPlayer.html
[5]http://www.javazoom.net/javalayer/javalayer.html

6.3 Testing Environment

Before the software MobileSOM is able to run on a mobile device, some requirements need to be met first.

6.3.1 Mobile Device Capabilities

In case MobileSOM will be installed on a real world device the MIDlet Suite "TestHandyAPIs.jar"[6] is used to test the connectivity and multimedia capabilities of the device. Without the ability to connect to a network, the software can neither be used as a remote control nor can music files be streamed from a server. To check connectivity the MIDlet "Internet" requests a response from a user defined URL and displays the result on the device. In case of success the response is printed in plain html. Otherwise an error message with a detailed description is shown. If the mobile device is used as a music player, it must be able to play audio files. Most devices support native audio formats like .wav or .au in Java, but not all of them support playback of mp3 files. To find out about the multimedia capabilities the MIDlet "PlayMP3" tries to play an mp3 file, that is included as a resource within the jar file. If the piece of music can not be played an error message is displayed.

6.3.2 Demonstration Music Libraries

To demonstrate the functionality of MobileSOM two music libraries are used. In the first library "Album Snippets" are 16 albums of various artists with a total number of 208 songs. The second audio collection "GTZAN" was used by George Tzanetakis in his master thesis [Tza02]. It consists of 1000 songs equi-distributed among 10 popular music genres. In Appendix A.3 the content and its visualisation (Map Images) are described. The demonstration package "MobileSOM.rar"[7] contains the "Album Snippets" library with 20 seconds mp3 snippets sampled in 48kbits/mono and ten prototypes 15 seconds long from the GTZAN library sampled in 32kbits/mono. The package also includes the Map Sources, the Map Images and the configuration files (see Section 6.1.1) for the two libraries.

[6] http://www.ifs.tuwien.ac.at/mir/pocketsom/mobilesom/dist/TestHandyAPIs.jar
[7] http://www.ifs.tuwien.ac.at/mir/pocketsom/mobilesom/dist/MobileSOM.rar

6.4 Installation

MobileSOM can either be installed on an emulator or on a real world device. This section guides through an installation process for running the software on a Sony Ericsson M600 Emulator followed by a section that shows how to install MobileSOM on real Nokia 7710.

6.4.1 Emulator

In order to run the Sony Ericsson M600 Emulator following system requirements have to be met:

Operation System : Microsoft Windows 2000/XP

Required hardware & memory [8] :

- 140 MB hard disk
- 256 MB system RAM
- 500 MHz CPU

Download of the :

- Sony Emulator Basic Package[9]
- Sony Emulator Add-on Package (5 & 6)[10]
- MobileSOM.jar file[11]
- MobileSOM.jad file[12]

The packages are installed on the machine. Before the emulator is started a default device has to be selected. This is either done from the start menu by clicking the "Default Device Selection" link (see Figure 6.7) or from the command line[13]. The emulator for the device "SonyEricsson_M600_Emu" is chosen from the drop down menu and the selection is confirmed by pressing the "ok"-button (see left screenshot in Figure 6.8). Now the emulator

[8] http://developer.sonyericsson.com/site/global/docstools/java/p_java.jsp

[9] http://www.ifs.tuwien.ac.at/mir/pocketsom/mobilesom/dist/Sony_Ericsson_Emulatoren/semc_java_me_sdk.2-2-3.exe

[10] http://www.ifs.tuwien.ac.at/mir/pocketsom/mobilesom/dist/Sony_Ericsson_Emulatoren/semc_java_me_sdk[1].2-2-3-addon5.exe
Sony_Ericsson_Emulatoren/semc_java_me_sdk[1].2-2-3-addon6.exe

[11] http://www.ifs.tuwien.ac.at/mir/pocketsom/mobilesom/dist/MobileSOM.jar

[12] http://www.ifs.tuwien.ac.at/mir/pocketsom/mobilesom/dist/MobileSOM.jad

[13] <program folder>\SonyEricsson\J2ME_SDK\PC_Emulation\WTK2\bin\DefaultDevicew.exe

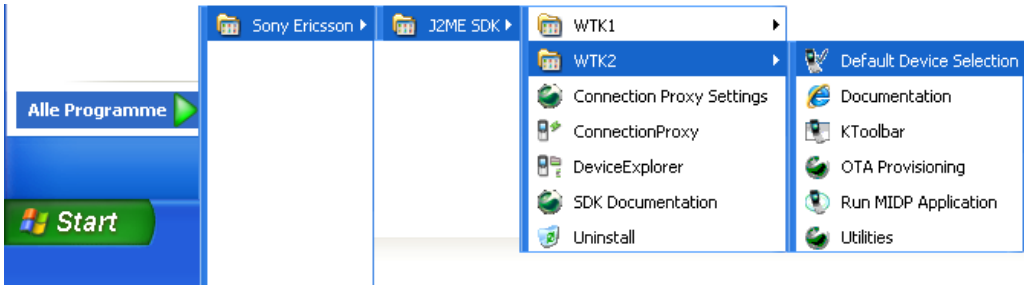

Figure 6.7: Start the device selection tool from the start menu

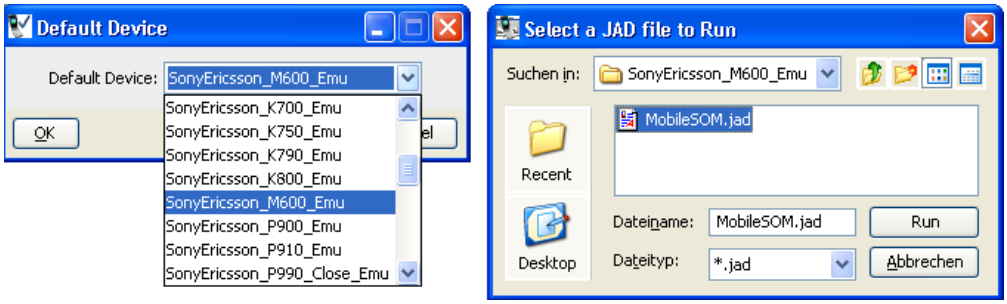

Figure 6.8: Left: Select a default device for emulation; Right: Select the MIDlet "MobileSOM" to run on the emulator

is started from the "Run MIDP Application" link in the start menu (see Figure 6.7) or from the command line[14]. A dialog opens up and asks for a .jad file to be executed. Now the downloaded file "MobileSOM.jad" is selected and after confirming, an emulation of the Sony Ericsson M600 running MobileSOM is shown (see in Figure 6.9). The prototype runs in demonstration mode by default, i.e. no songs are played. To change this behaviour the MIDlet "Load Customized Settings" is opened and from the predefined settings the last entry[15] is selected like shown in Figure 6.9. MobileSOM connects to the Internet and downloads a Map Image[16] showing a 4x4 Grid with 16 album covers, the Map Source[17] assigning 208 songs to the map and sets up the Mp3 Server[18] offering mp3 snippets 20 seconds long and sampled in 48kbits/mono. The Map Image, the Map Source and the settings are stored to the RMS of the device which in case of the emulator is a directory

[14] <program folder>\SonyEricsson\J2ME_SDK\PC_Emulation\WTK2\bin\emulatorw.exe -gui -Xdescriptor:

[15] http://www.ifs.tuwien.ac.at/mir/pocketsom/mobilesom/soms/samples.som

[16] http://www.ifs.tuwien.ac.at/mir/pocketsom/mobilesom/soms/samples.jpg

[17] http://www.ifs.tuwien.ac.at/mir/pocketsom/mobilesom/soms/samples.list

[18] http://www.ifs.tuwien.ac.at/mir/pocketsom/mobilesom/muke/samples/48/

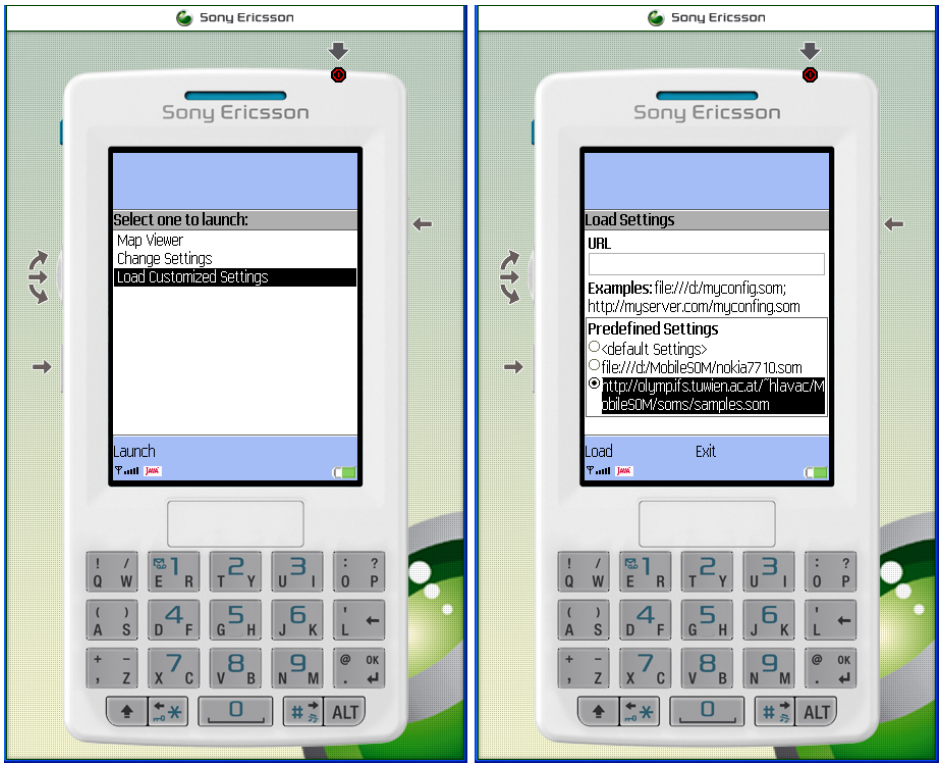

Figure 6.9: Left: The welcome screen of the MobileSOM Suite; Right: The "Load Customized Settings"-MIDlet with a predefined setting selected.

on the hard disk[19]. The first attempt from any MIDlet trying to connect to the network has to be confirmed by the user one time. The software is now in live mode and the "Map Viewer"-MIDlet may be started to interact with the Map Image.

6.4.2 Real World Device: Nokia 7710

This section describes the installation of MobileSOM on a Nokia 7710. To transfer MobileSOM to a Nokia device, the Nokia PC Suite[20] needs to be installed first. Following system requirements have to be met:

Operation System : Windows 2000 (Service Pack 4), Windows XP SP2

Connectivity : USB cable, Infrared or Bluetooth

[19] <program folder>\SonyEricsson\J2ME_SDK\PC_Emulation\WTK2\appdb\SonyEricsson_M600_Emu

[20] http://www.nokia.de/de/service/software/pc_suite/download/114962.html

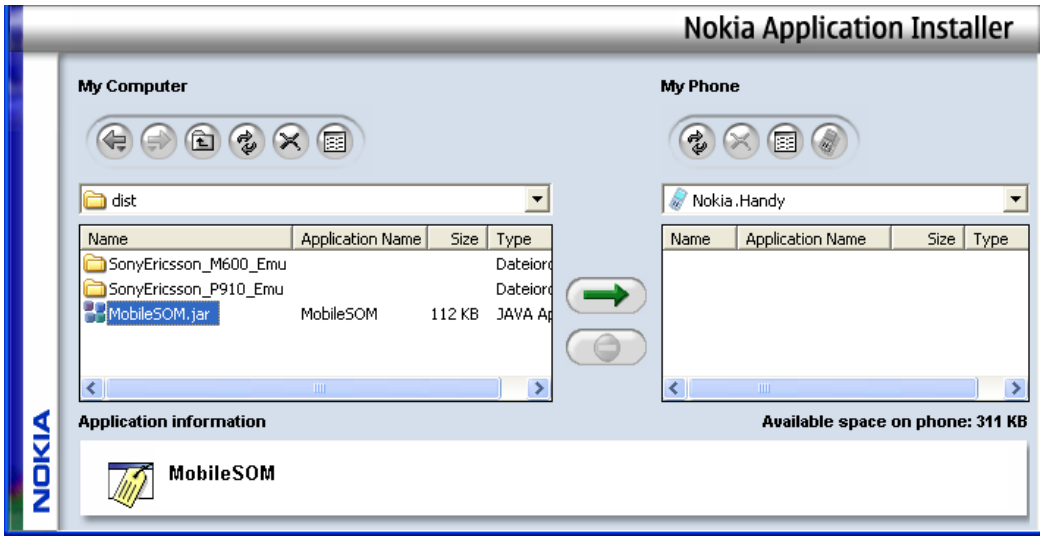

Figure 6.10: Nokia Application Installer: Send a MIDlet to a Nokia Device

Required hardware & memory :

- 200 MB free hard disk space
- 256 MB system RAM
- 500 MHz CPU

Now the "MobileSOM.jar" file[21] has to be downloaded. A double-click on the file opens the Nokia Application Installer (see Figure 6.10). After establishing a connection to the Nokia 7710, the file will be transferred by selecting it and clicking the button with the green arrow next to it. The rest of the installation process is done on the device itself by confirming all message boxes that show up. In the next step the demonstration package "MobileSOM.rar"[22], which contains the music files, is download and unzipped. With the help of the PC Suite the extracted folder "MobileSOM" is moved to the memory card (drive letter "d:") of the mobile device. On the Nokia phone the "Load Customized Settings"-MIDlet is double clicked and from the "Predefined Settings" menu the second entry[23] is loaded. During the loading process the Map Image and the Map Source are transferred to the RMS and the local mp3 directory is set to the directory on the memory card[24] where the music snippets are stored. Now the "Map Viewer"-MIDlet may be started to

[21] http://www.ifs.tuwien.ac.at/mir/pocketsom/mobilesom/dist/MobileSOM.jar
[22] http://www.ifs.tuwien.ac.at/mir/pocketsom/mobilesom/dist/MobileSOM.rar
[23] file:///d:/MobileSOM/nokia7710.som
[24] file:///d:/MobileSOM/pezzo/

Figure 6.11: A screenshot of MobileSOM running on a real world Nokia 7710

interact with the Map Image (see Figure 6.11). **Note:** Because the MIDlet suite is not signed, every files access has to be confirmed manually by the user. In case of a network access the user has to confirm only once.

Other Devices

The installation of MobileSOM on other mobile devices may work directly from the internal web browser. For example using the Qtek 9100 the software is downloaded and installed to the device by calling the URL of the jar file in the address bar of the browser.

6.5 Configurations

MobileSOM can either run in

- 0 Demonstration Mode,
- 1 Live Mode or
- 2 Remote Mode.

Between Live and Remote Mode is switched in the "Map Viewer"-MIDlet by pressing the key "1" or in the "Change Setting Viewer"-MIDlet by choosing any of the three modes. The Demonstration Mode is for user interface testing but does not play any music files. In Live Mode music files are streamed from the network or played back from the local memory of the device. In Remote Mode pieces of music are played on a different machine which is

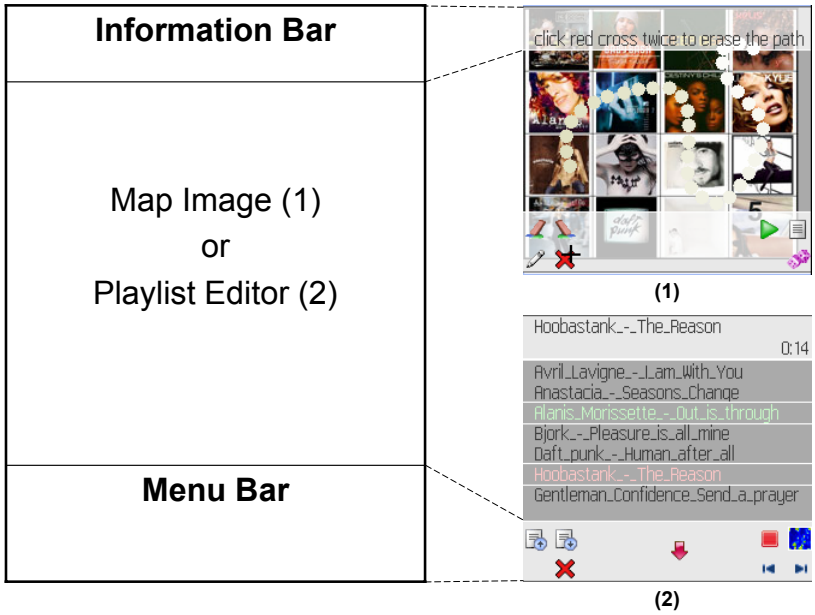

Figure 6.12: The User Interface partitioned into an Information Bar, the Map Image or the Playlist Editor and a Menu Bar

remote controlled by the MobileSOM. Map Images and Map Sources are either downloaded from the Internet or taken from the local file system of the device and can be exchanged during run-time in the "Change Setting Viewer"-MIDlet. The local mp3 directory and the mp3 server are set to any local or web folder containing the songs listed in the Map Source.

6.6 User Interaction

6.6.1 The User Interface

The user interface (Figure 6.12) of the "Map Viewer"-MIDlet is divided into three parts:

- a semi-transparent Information Bar placed on the top of the screen
- a display filling Map Image or the Playlist Editor
- a semi-transparent Menu Bar placed at the bottom of the screen

The Information Bar is able to display two lines of text (e.g. tooltips or song names). The number of characters per line depends on the default

font size and the pixel resolution of the screen, which is different on every device. If a message is too long and therefore would be truncated a news ticker algorithm was implemented that smoothly scrolls the message from left to the right and vice versa. The Menu Bar shows icons that allow the user to switch between different interaction modes. Like discussed in [JM06], see chapter 8.4, it is hard to pick an icon that unambiguously represents a function. A solution is to display icons in combination with text to reinforce the idea. Because of limited screen size, the icons are used without text in MobileSOM. They are explained by the Welcome Assistant (see section 6.6.4) that shows a short description in the information bar the first time an icon is displayed. The Map Image is displayed on the whole screen. From the Map Image pieces of music are selected by the user and may be rearranged in the Playlist Editor.

6.6.2 Using the Stylus

For convenient user interaction the software responds to any action taken by the user within a very short time. This ensures that the user is always aware about any state change or what is going to happen next. A simple feedback mechanism is that any click with the stylus on a specific location on the screen produces a small circle on that place which fades out within a half second (Figure 6.13(1)). The user knows immediately, if the click was registered by the software and where exactly the stylus was placed on the screen. With a click on the device either a menu icon is pressed or a piece of music is selected. For selecting multiple music items the stylus is placed on the Map Image and a path is drawn over the screen as long the stylus is not taken off (Figure 6.13(2)). Another interaction method is to mop or circulate the stylus over a specific menu icon (Figure 6.13(3,4)). This simulates multiple clicks on that icon without the need of removing the stylus from the display. For example the more mopping the stylus on the rubber icon, the more the path drawn by the user is shortened.

6.6.3 Hotkeys

Even if all features in the “Map Viewer”-MIDlet can be accomplished with the stylus, some hotkeys enhance the usability of the software like pressing

- “1” to switch between Live- and Remote Mode
- “2 or 3” to change the volume of the playback
- “5” to activate or deactivate Drawing Mode

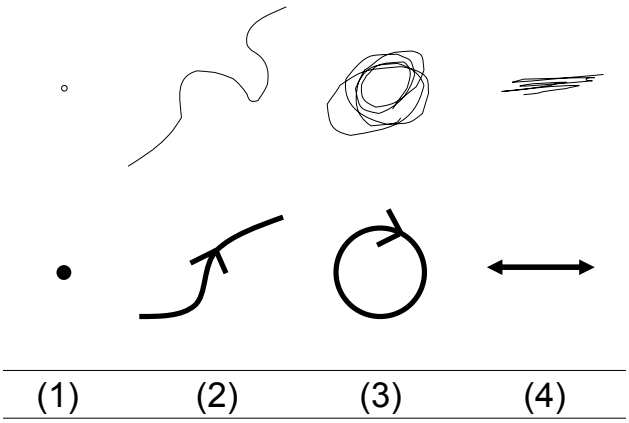

Figure 6.13: Stylus Moves: (1) clicking, (2) drawing, (3) circulating, (4) mopping

- "Up", "Down", "Left" & "Right" to scroll Map Image

6.6.4 Welcome Assistant

The first time the "Map Viewer"-MIDlet is started an assistant will guide the user through the first interaction steps. On the screen the Map Image and two white semi-transparent bars are shown. In the information a suggestion for moving the stylus on the display is displayed to the user (Figure 6.14(1)). As soon the stylus is tabbed on the screen and moved around, the top bar shows the name of the underlying items for every Unit (Figure 6.14(2)). If there are many items on a Unit, the item names are displayed by circulating the stylus on that Unit. After deducting the stylus, the assistant suggests the user to click on the pencil icon at the left bottom to draw a path on the map (Figure 6.14(3)). If the pencil is clicked, the desaturated icon turns red and the assistant informs the user that it is possible now to draw a path on the map (see Figure 6.14(4)). If the stylus is moved, a path consisting of white filled circles is painted (see Figure 6.14(5)). The longer the path gets, the more the circles fade from the top of the path to the tail to a darker colour tone. This history function ensures that the user is always aware, where he tabbed the stylus for the last time. If drawing the path is done, the assistant suggests to click on the white list icon at the right bottom (Figure 6.14(6)) to show the path's underlying items Figure 6.14(7). The playlist shown in this figure is built the following way: Every circle from the path is painted somewhere over a unit (in this case a unit is represented as an album cover).

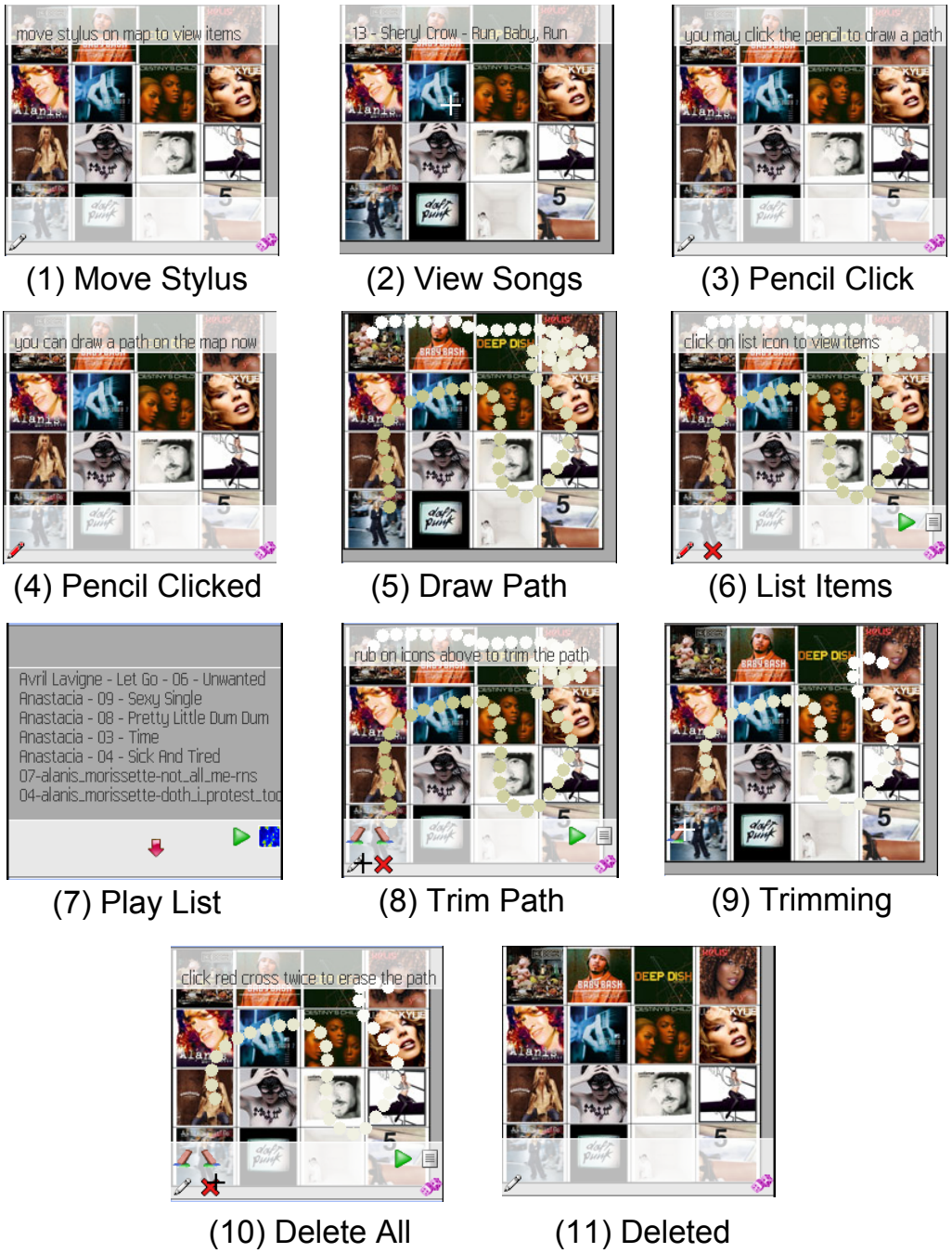

Figure 6.14: Welcome Assistant

From that unit one song is taken and added to the playlist. If more circles are painted on one unit, different tracks are chosen from it. In case there are more circles painted on a cover than songs are available, no further action is taken (no item is added twice to a playlist by design). The user can either listen to the songs or edit the playlist on the map or in the Playlist Editor (see section 6.6.7). On the map the path is trimmed by rubbing the icons above the pencil (Figure 6.14(8)). The left rubber cuts away circles from the tail and the other one from the head. The trimmed path is shown in Figure 6.14(9). During trimming the pencil has to be deactivated. A playlist is deleted by clicking the red cross next to the pencil. After tabbing the icon one time, the assistant tells the user to click the red cross twice to delete the entire path (see Figure 6.14(10)). This function is implemented for security reasons as not to erase the whole playlist by accident.

6.6.5 Roll the Dice

If a path is drawn over an album, each circle selects a song from the album. The way the songs are added to the playlist is either sequentially or by random. If the "Roll the Dice"-function is deactivated, the first circle painted on the cover adds the first song of the album to the playlist, the second circle the second song of the album and so on. So any time a path is drawn over an album cover the first songs of the album are taken and the user might never listen to pieces of music placed at the end of the album. If the "Roll the Dice"-function is activated, the first circle painted on the cover adds a random selected song of the album to the playlist, the second circle another random selected song of the album that is not already in the playlist and so on (see Figure 6.15). The advantage of the function is that playlists generated by the user will differ every time, even if they draw exactly the same path. To enable the "Roll the Dice"-function the dice icon at the bottom right of the menu bar is clicked until is pink coloured and deactivated by tabbing on it again so the colour of the icon will change back to gray.

6.6.6 Scrolling and Zooming

The Map Image can have any desired dimensions as long the file size does not exceed the memory capabilities of the mobile device. If the map is clipped by the border of the display, navigation arrows will blend in the menu bar. The user scrolls the map either by clicking the red arrows in the middle of the menu bar or moves the stylus to one of the four regions close to the red highlighted borders in Figure 6.16. A feature that was used in an earlier version of MobileSOM is zooming in the map. With a magnifier icon for

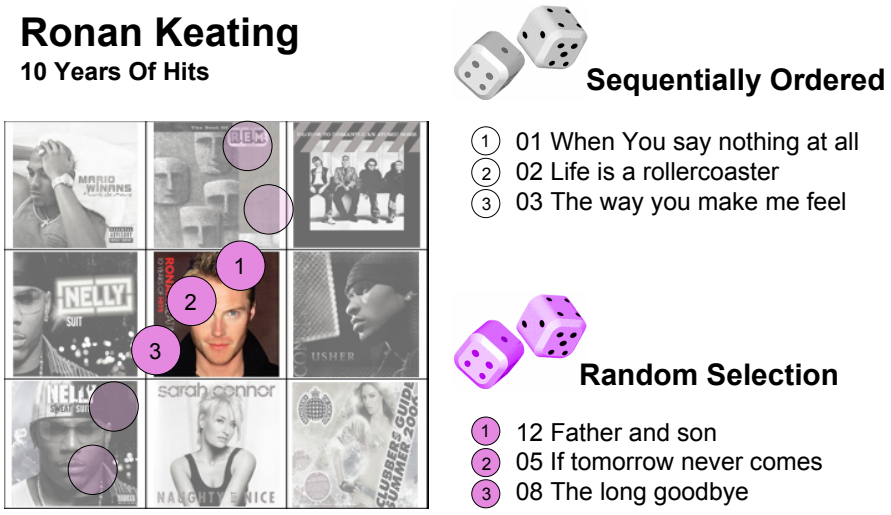

Figure 6.15: "Roll the Dice"-function: Songs are chosen from an album randomly or in a sequentially order

zooming in and out it was possible to switch between four different levels of zoom. Because of the fact that the whole Map Image was decoded into an integer array, on most mobile devices a heap overflow occurred and the device had to be rebooted. The function was impracticable on real world devices and was therefore removed.

6.6.7 Playlist Editor

In the Editor all songs that where selected are displayed in a list view (see Figure 6.17). If the list is to long, scrolling icons appear in the menu bar recognizable as red arrows. For fast scrolling the stylus can be mopped on one of the arrows instead of singly pointing on them. The created playlist can be altered by removing songs or changing their position. To perform operations on an item, it is selected first by clicking on it. The selected item is now highlighted light green. By clicking the red cross icon, the song is removed from the playlist and its successor is selected. To move a song up or down in the list, one of the list icons in the upper left menu bar is clicked. For quickly moving an item through the list the stylus is mopped on one of the list icons. Mopping the stylus simulates multiple clicks on the screen. To

Figure 6.16: Scrolling and Zooming

Figure 6.17: Playlist Editor

Figure 6.18: Editing features from left to the right: Red arrows for scrolling up and down in the list, list items for moving a selected song up or down, rubber for deleting a single item, play/stop button for playing items from the playlist, blue arrows for skipping tracks, map icon to switch to the map interface

switch back to the Map Image the map icon on the right is clicked.

6.6.8 The Player

The Player is located as a transparent bar at the top of the screen and can be activated as soon as any songs are added to the playlist by clicking the play icon either on the Map Image or in the Playlist Editor. The name of the current played song and the time progress are displayed in the player window. In Playlist Editor the actual played song is highlighted pink. Titles are skipped by clicking on the forward or backward icon. The player is stopped by clicking on the stop button. There are three player modes:

1. Play files from the local file system
2. Stream files from a server
3. Play files on a remote machine

In the first case all pieces of music are available on the mobile device and are played locally. The other modes require a network connection. An issue to overcome is that streaming of audio files is not supported by the MMAPI. The files are first downloaded to the device and played again locally. This results in long waiting times which is very obvious when the media file is big. A special feature for devices that are not able to play audio files is the remote mode. Music files are stored on a PC that is connected to a network and a server application is waiting for incoming item requests. After creating a playlist on the mobile device, the device acts as a remote control telling the server which song to play. It is even possible to switch between remote and local mode at runtime. An “R” placed at the right top of the play icon indicates that the song is played on another device remotely. The remote mode is switched on or off by pressing key “1” on the device. Figure 6.19 shows the player in the two different modes.

Figure 6.19: Player in remote mode on the left and playing music from the mobile device on the right side

Figure 6.20: Deployment of a MIDlet

6.7 Emulators versus Real World Devices

The prototype was developed in Netbeans which is an open source Java IDE that provides creating mobile applications straight out of the box. Once a MIDlet is ready to distribute, it is tested on an emulator first. Netbeans supports direct integration of emulators by various manufacturers. MobileSOM was tested on the following emulators

1. Prototype 4.0 Nokia 7710 MIDP Emulator
2. Sony Ericsson MIDP 2.0 and CLDC 1.1 P900/P910 Emulator
3. Sony Ericsson MIDP 2.0 and CLDC 1.1 M600 Emulator

Moreover the IDE supports developing applications for multiple devices by adding and executing device-specific code as configurations within a single MIDlet, called "device fragmentation". If the program passed all functional tests in the emulator environment there are several ways to deploy it on a device for example via Cable, Bluetooth or IR, over WAP-Push or download it directly from a server. Figure 6.20 shows the deployment from a MIDlet to a Nokia device. Unfortunately the deployment procedure described above turned out to be impracticable in many aspects. The MobileSOM Player loads many Images at the beginning, which can result in an *out of runtime memory error* on real devices. A cold restart of the device is necessary. Running out of heap memory (=runtime memory), has nothing to do with the

maximal size of a MIDlet suite (=jar File). For example, a compressed image might only take 6K in the JAR file, but it uses about 10 times more heap space during runtime. Pictures take about 2 byte/pixel memory regardless of file size. The fact that this error occurs is a sign that the software is trying to allocate more memory than is available as a single chunk. This usually occurs when a program attempts to load a large image after loading many smaller ones. The smaller images used up half the heap and the large one can not be provided a large enough chunk in which to fit. In general a program must avoid loading such large resources into memory. The only way to tell whether a MIDlet is too big to run on the phone is to test it on the actual device. Emulators cannot be accurate in simulating the heap memory management.

The Multi Media API (MMAPI) is implemented differently by every manufacturer. Most of the mobile device do not support mp3 playback at all. The most complete implementation of the JSR-135 can be found on various Sony Ericsson devices (e.g. M600 or P910) followed by some Nokia mobiles. Playing mp3's using the MMAPI works on a Nokia 7710 but it does not work on its Prototype 4.0 Nokia 7710 MIDP Emulator.

Lemma 1 (Deployment) *If a MIDlet works on the emulator, it may not work on the device.*

Lemma 2 (Deployment) *If a MIDlet works on the device, it may not work on the emulator.*

6.8 Practical Experiences on Real World Devices

An evaluation of the MobileSOM running on a Nokia 7710 and a Sony Ericsson P910 is given. Because of heap memory limitations the zooming feature is disabled. Due to the fact that streaming is performed in downloading the songs first to the device, which leads into long waiting times, all devices stored the music content locally. The audio archive used contains 10 Songs from 10 genres. After installing MobileSOM on both devices the first observed difference is how a MIDlet Suite is organised on the device. On the Nokia 7710 every MIDlet of the suite is shown as a labelled icon in the program manager ordered by its names. If there are more MIDlet Suites installed, the user will lose track of which MIDlet belongs to which suite. On the P910 MIDlet Suites are displayed as folders containing the MIDlets. If the user opens a suite he can choose in a list box between the available MIDlets.

6.8.1 Interaction with the Stylus

In this section a closer look at the "Map Viewer"-MIDlet considering user interaction with the stylus is taken. Clicking menu icons is an easy task on the emulators but could be a tricky one on all real world devices because of calibration problems or hitting wrong icons because they are too small. It has to be ensured that the icons are big enough and well separated. Because of different screen resolutions the size of the menu items and the distance between them has to be adjusted for every device. Drawing a path is smooth on all Sony Ericsson devices but a fatiguing task on the Nokia 7710. The graphical computation power of the Nokia 7710 is much weaker than on Sony Ericsson ones both on the emulators and the real world devices. On Sony Ericsson devices the path is drawn as soon the screen is touched with the stylus. On the Nokia 7710 is a noticeable delay between tabbing the screen until the path is drawn on the device. This delay increases as the path gets longer. Once a path is drawn, the rubbing effect is tested. Mopping the stylus over the rubber icon feels like erasing a painting with a real rubber. Using the stylus like an "ink-killer" is much more intuitive then using the mouse for the same task on the emulator.

6.8.2 Multimedia

The next section concentrates in the multimedia capabilities of the tested devices. Excluded is the Prototype 4.0 Nokia 7710 MIDP Emulator because of the lack of mp3 support. Songs start to play on the emulators instantly but with a noticeable delay on the real world devices. The initialization process to play an mp3 on real world devices takes about one to four seconds. Using the mobiles as a remote control works fine on all devices. As soon they were connected to a network, remote commands worked with almost no noticeable delay.

6.8.3 Sandbox

The sandbox is a set of rules that are used when creating an MIDlet that prevents certain functions when the MIDlet is running on a mobile device. It creates an environment in which there are strict limitations on what kind of system resources the MIDlet is allowed to access. Sandboxes are used when executable code comes from unknown sources thus allow the user to run untrusted code safely. MobileSOM is recognized as an untrusted source by every real world device, because it is not signed. For the first time MobileSOM wants to access a system resource like the local file system, the

network or the RMS, the user has to confirm the request. The security settings on a Nokia 7710 are even stricter. Every access to the file system has to be confirmed by the user. This makes it impossible to listen to a generated playlist without interrupting the playback. After a song finished the user has to acknowledge the access to the file system for the following track again. MobileSOM has not been signed yet, because the costs for signing a MIDlet bear no relation to software that is still in a prototype state.

6.9 Future Extensions

In this section challenging ideas for future extensions of MobileSOM are given.

6.9.1 Switching between different Representation Layers

The current version of MobileSOM implements a zooming algorithm, that interpolates a given image to an arbitrary size. Because of heap limitations on real world devices it is not possible to scale the Map Image during runtime which would lead into a buffer overflow. Another approach to overcome this problem is to split the Map Image into Unit Images. A Unit Image is saved in different sizes for every zooming layer. Depending on the part of the map that is viewed, the corresponding Unit Images are shown on the screen. Having images of different size for every Unit makes it easy to realise the concept of growing hierarchical Self-Organizing Maps[RMD02], where from a Unit that has more items than a given threshold a new SOM is computed. At a certain zooming level the graphical visualisation of a Unit could be combined with a textual representation of the corresponding map items. This was done in [NDR05] where the names of all songs were displayed in a unit.

6.9.2 Drag & Drop Functionality

So far MobileSOM does not offer any interaction facilities to manipulate the Map Image or to rearrange the Map Items on the Grid. A way of reorganising a map is to exchange Units. This could be done by holding the stylus on a specific Unit for about two seconds to activate the drag mode. Now the Unit is dragged over the map and dropped over any other Unit. In the same way Map Items can be moved from one Unit to another.

6.9.3 Mouse Pointer

The current version of MobileSOM requires a device with touchscreen in order to draw a trajectory on the display. An implementation of a "mouse pointer" which is controlled with the keys up, down, left and right would enable the entire controll of the prototype with the keyboard and would make the application available for a wider range of mobile devices.

6.9.4 Extended Playlist Generation

As described in Section 6.1.4 the algorithm for generating a playlist takes a fixed number of songs from every slice of the path and adds them to the playlist. A different approach is to let the user define a desired duration of the playlist before any items are selected. After a path is drawn an appropriate number of songs is taken where the sum of the durations matches the user defined length.

6.9.5 Progressive download

With progressive download, playback can start as soon as a certain amount of data has been buffered in the phone memory. This allows the user to listen to a song while it is still being downloaded. The current version of MobileSOM has to download the whole song to the device before the playback begins. With newer devices, that implement this technique already in the JVM[25], this problem will be solved.

6.9.6 On-Device Feature Extraction & Map Image Re-design

With increasing memory capabilities and faster computation power on mobile devices it will be possible to extract features of audio files on the device itself. Similarity matrices or Map Images are created during runtime without the need of external hard- and software. MobileSOM would act as a complete independent software package, which is able to deal with growing music libraries and dynamically organising them on the device.

6.10 Summary

In this chapter the prototype MobileSOM and the XPlayer were introduced. The reader learned about the software architecture and the environment that

[25] http://developer.sonyericsson.com/site/global/techsupport/tipstrickscode/java/p_new_features_mobilemedia_api_jsr135.jsp

is necessary in order to run MobileSOM on a mobile device. An installation guide showed how to deploy the prototype on a emulator and on a real world device. In the next sections the reader learned about the configuration possibilities and how to interact with the prototype. A comparison of running MobileSOM on a emulator and real world device was given and practical experiences of using the prototype on a Nokia 7710 and on a Sony Ericsson P910 were discussed. In the end of this chapter ideas of how to extend MobileSOM with new features were given.

Chapter 7
Conclusion

In this chapter the work of the thesis is summarized and in addition, opportunities for future research and further development of the presented prototype are discussed.

7.1 Innovative User Interfaces for accessing Music on Mobile Devices

The work presents innovative concepts for visualizing and interacting with music libraries on mobile devices. The basic idea is to display a music archive as a two-dimensional map. The map is split into units that contain pieces of music with similar properties, for example a unit contains music from a specific artist or genre. Even more abstract similarities like songs that sound similar or pieces of music of the same mood can be identified using feature extraction techniques. To arrange units in the grid a self-organizing map algorithm is applied which places units that contain similar songs next to each other on the map. Units can be visualized as album covers or emoticons representing a specific mood. A more abstract visualization type is to use a topographic map which displays units with a high density of songs as islands and units with fewer pieces of music as water.
The concepts were realized in a mobile application called **MobileSOM** (Mobile Selection of Music) using Java Micro Edition (J2ME) as programming language. In order to run MobileSOM, the software needs a map image, which visualizes the music archive and a data file, which contains the path to the music files and the location of them on the map, as input parameters. During runtime songs can be selected from the music map and played on the device. The prototype can also be used as a remote control. Instead of playing a piece of music on the mobile device MobileSOM triggers another device to play the song.

Today only a limited number of real world devices have enough computation power and multimedia capabilities to run MobileSOM in its full functionality. For demonstration purposes a small set of state-of-the-art devices were carefully selected from various manufacturers to run a slimmed version of MobileSOM. Testing the prototype on real world devices showed that the

described features are practicable. They will perform smoother if the hardware becomes more powerful and the network connections for accessing music content get faster.

MobileSOM has not reached a level, that suggests its commercial usage. Still, devices of the next generations will overcome the technological bottle neck so that this work will be a great inspiration for commercial music applications on mobile devices in the near future.

7.2 Future Work

The demonstration music libraries used in MobileSOM contained less than thousand songs. An interesting task will be to investigate the performance and usability of MobileSOM working with huge audio collections. Another issue is to find out if the concept of music maps is suitable for commercial audio archives containing millions of songs or if MobileSOM works better with private audio collections. A challenging task will be the integration of feature extraction techniques and the implementation of the self-organizing map algorithm on mobile devices. MobileSOM would then be able to place new songs to corresponding units on the map without the need for external soft- or hardware.

Appendix A
Appendix

A.1 Build a Map Item List

```
1  /**
2   * Builds a MapItemList from the path, that was drawn
3   * by the user.The path consists of Map Points. Every
4   * Map Point belongs to a Unit. From that Units Map
5   * Items are taken and added to the MapItemList. Map
6   * Items are choosen from a Unit either randomly or in
7   * a ordered manner from the first to the last item.
8   * <br>
9   * An example: <code>build</code> is called with <code>
10  * bRandomize</code> set to <code>false</code> and
11  * <code>maxNumberOfMapItems</code> set to 3. For a Map
12  * Point placed in Unit X the first 3 Map Items of that
13  * Unit are added to the MapItemList.<br>
14  * The MapItemList never contains any Map Item twice.
15  *
16  * @param mapPoints the list of Map Points
17  * @param grid the Grid containing Map Items
18  * @param bRandomize if set to <code>true</code> Map
19  * Items from a Unit are selected randomly.
20  * @param maxNumberOfMapItems the maximum number of Map
21  * Items taken from a Unit
22  */
23 public void build(
24          Vector mapPoints,
25          Grid grid,
26          boolean bRandomize,
27          int maxNumberOfMapItems)
28 {
29
30    Vector    itemList;
31    MapPoint  mapPoint;
32    MapItem   MapItem;
33    int       itemCounter = 0;
```

```
34
35    this.removeAllElements();
36
37    // go through the points (mapPoints) that the user
38    // painted on the screen
39    for ( Enumeration e = mapPoints.elements() ;
40                          e.hasMoreElements() ;)
41    {
42                  mapPoint = (MapPoint) e.nextElement();
43
44      // a Map Point has a corresponding Unit.
45      itemList = grid.getMapItems(
46                            mapPoint.getUnitId());
47
48      // if there are Map Items in that Unit, add them
49      // to the MapItemList(there is a fixed number of
50      // items per Map Point (MAX_SONGS_PER_MAPPOINT)
51      // that will be added to the MapItemList)
52      itemCounter = 0;
53
54      if (itemList!=null)
55      {
56        // go through the items from the actual unit and
57        // if they are not already in the playlist add
58        // them to it.
59        if (bRandomize)
60        {
61          // get randomized Map Items, if you get bored
62          // listening always to the same pick one random
63          // item from itemList and add it to the
64          // playlist. Repeat until itemlist is empty or
65          // more than "maxNumberOfMapItems" where picked.
66          int n;
67          if (itemList.size()>0)
68          {
69            n = Math.abs(random.nextInt())
70                % itemList.size();
71
72            MapItem = (MapItem)itemList.elementAt(n);
73
74            if (!this.contains(MapItem))
```

```
75              {
76                 this.addElement(MapItem);
77              }
78              itemCounter++;
79           }
80        }
81        else
82        {
83           for (Enumeration i=itemList.elements();
84                 i.hasMoreElements();)
85           {
86              MapItem = (MapItem)i.nextElement();
87
88              if (!this.contains(MapItem) &&
89                    itemCounter < maxNumberOfMapItems)
90              {
91                 this.addElement(MapItem);
92                 itemCounter++;
93              }
94           }
95        }
96     }
97  }
98 }
```

A.2 Device Specifications

Nokia 7710	
OS	Symbian OS v7.0s, Series 90 UI
Processor	ARM based processor 150 MHz
Interactivity	TFT touch-screen 65K colours 640 x 320 pixels
Memory	Card slot MMC, 128 MB card included 90 MB shared internal memory
Connectivity	Data GPRS Class 10, 32 - 48 kbps HSCSD, 43.2 kbps EDGE Class 10, 236.8 kbps No WLAN Bluetooth, v1.2 No Infrared port USB
Customization	Java MIDP 2.0, CLDC-1.1 JSR 135 (MP3 decoding implemented)

Figure A.1: Nokia 7710 Specifications

Sony-Ericsson M600

OS	Symbian OS v9.1, UIQ 3.0
Interactivity	TFT touchscreen 256K colors 240 x 320 pixels QWERTY keyboard
Memory	64 MB card included 80 MB shared memory
Connectivity	GPRS Class 10, 32 - 48 kbps HSCSD 43.2 kbps No EDGE 3G Yes, 384 kbps No WLAN Bluetooth v2.0 Infrared port USB v2.0
Customization	Java MIDP 2.0, CLDC-1.1 JSR 135 (MP3 decoding implemented)

Figure A.2: Sony Ericsson M600 Specifications

Qtek 9100

OS	Microsoft Windows Mobile 5.0 PocketPC
Processor	TI OMAP 850, 200 Mhz processor
Interactivity	TFT touch-screen 65K colours 240 x 320 pixels QWERTY keyboard
Memory	64 MB DDR SDRAM 128 MB ROM
Connectivity	GPRS Class 10, 32 - 48 kbps EDGE Class 10, 236.8 kbps WLAN Wi-Fi 802.11b/g Bluetooth v2.0 Infrared port USB
Customization	Java MIDP 2.0, CLDC-1.1 JSR 135 (MP3 decoding implemented)

Figure A.3: Qtek 9100 Specifications

BenQ P50	
OS	Microsoft Windows Pocket PC 2003 Phone edition
Processor	Intel PXA 272, 416 MHz
Interactivity	TFT touch-screen 65K colours 240 x 320 pixels
Memory	64 MB SDRAM 64 MB Flash ROM
Connectivity	GPRS Class 10 (4+1/3+2 slots), 32 - 48 kbps WLAN Wi-Fi 802.11b Bluetooth, v1.1 Infrared port USB
Customization	Java MIDP 2.0, CLDC-1.0 JSR 135 (MP3 decoding implemented)

Figure A.4: Benq P50 Specifications

A.3 Music

Songs	**Album**	**Artist**
12	3 Doors Down	Seventeen Days
10	Alanis Morissette	So Called Chaos
12	Anastacia	Anastacia
14	Avril Lavigne	Let Go
3	Baby Bash	Suga Suga
16	Best of MTV	Unplugged Vol.2
12	Bjork	Medulla
9	Daft Punk	Human After All
14	Deep Dish	George Is On
12	Destinys Child	Destiny Fulfilled
20	Gentleman	Confidence
14	Hoobastank	The Reason
14	Kelis	Tasty
33	Kylie Minogue	Ultimate Kylie
13	Lenny Kravitz	5
208		

Table A.1: Test set “Album Snippets”: 16 albums from various artists with 208 songs total

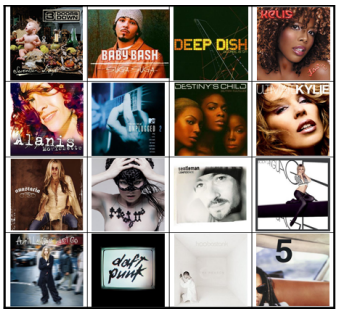

Figure A.5: Map Image showing album covers from table A.1 in a 4x4 Map Grid

Songs	Genres
100	Disco
100	Hiphop
100	Blues
100	Pop
100	Reggae
100	Metal
100	Jazz
100	Country
100	Rock
100	Classical
1000	

Table A.2: Test set “Gtzan”: 1000 songs divided into 10 genre à 100 pieces

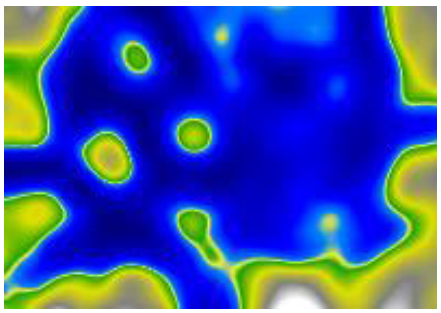

Figure A.6: Map Image showing a SOM (20x14 Units) computed with the test set from table A.2

Bibliography

[AP03] Jean-Julien Aucouturier and François Pachet. Representing musical genre: A state of the art. *Journal of New Music Research*, 32(1):83 – 93, March 2003.

[App] Apple. iTunes Store. http://www.apple.com/de/itunes/store/, accessed on 22nd of Nov.2006.

[Arn99] Michael Arnold. Mp3 robust audio watermarking. In *Proceedings of the DFG VIIIDII Watermarking Workshop*, Fraunhofer Institute for Computer Graphics IGD, Oct. 5-6 1999.

[Bau06] Doris Baum. Emomusic - Classifying music according to emotion. In *Proceedings of the 7th Workshop on Data Analysis (WDA)*, Kosice, Slovakia, July 1-3 2006.

[BR05] Thorsten Buering and Harald Reiterer. Zuiscat - Querying and visualizing information spaces on personal digital assistants. In *Proceedings of the 7th international conference on Human computer interaction with mobile devices & services (MobileHCI)*, pages 129–136, New York, NY, USA, Sep 2005. ACM Press.

[BR06] Doris Baum and Andreas Rauber. Emotional descriptors for map-based access to music libraries. In *Proceedings of the 9th International Conference on Asian Digital Libraries (ICADL)*, Kyoto, Japan, November 27-30 2006.

[Bra03] Karlheinz Brandenburg. Mp3 and Aac explained, 2003.

[dBS00] Oscar de Bruijn and Robert Spence. Rapid serial visual presentation: A space-time trade-off in information presentation. In *Proceedings of Advanced Visual Interfaces (AVI)*, Palermo, Italy, May 2000.

[DNR05] Michael Dittenbach, Robert Neumayer, and Andreas Rauber. Playsom: An alternative approach to track selection and playlist generation in large music collections. In *Proceedings of the First International Workshop of the EU Network of Excellence DELOS on Audio-Visual Content and Information Visualization in Digital Libraries (AVIVDiLib)*, pages 226–235, Cortona, Italy, May 4-6 2005.

[Eri] Sony Ericsson. Mobile music. http://developer.sonyericsson.com/site/global/newsandevents/campaigns/mobile_music/p_mobilemusic.jsp, accessed on 9th of Jan.2007.

[HFG+98] Beverly L. Harrison, Kenneth P. Fishkin, Anuj Gujar, Carlos Mochon, and Roy Want. Squeeze me, hold me, tilt me! An exploration of manipulative user interfaces. In *Proceedings of the conference on Human Factors in Computing Systems (SIGCHI)*, pages 17–24. ACM Press/Addison-Wesley Publishing Co., 1998.

[IFP07] IFPI. Digital music report. Technical report, IFPI, January, 17th 2007. `http://www.ifpi.org/content/library/digital-music-report-2007.pdf`.

[JM06] Matt Jones and Gary Marsden. *Mobile Interaction Design*. John Wiley & Sons, February 2006.

[Kob05] Evan Koblentz. The evolution of the PDA, May 2005. http://www.snarc.net/pda/pda-treatise.htm, accessed on 3rd of Dec.2006.

[Koh95] Teuvo Kohonen. *Self-Organizing Maps*. Springer-Verlag, Berlin, 1995.

[LO03] Tao Li and Mitsunori Ogihara. Detecting emotion in music. In *Proceedings of the Fourth International Conference on Music Information Retrieval (ISMIR)*, 2003.

[Mas98] Toshiyuki Masui. An efficient text input method for pen-based computers. In *Proceedings of the conference on Human Factors in Computing Systems (SIGCHI)*, pages 328–335, New York, NY, USA, 1998. ACM Press/Addison-Wesley Publishing Co.

[MC02] Michael Moyle and Andy Cockburn. Analysing mouse and pen flick gestures. In *Proceedings of the Symposium On Computer-Human Interaction (SIGCHI-NZ)*, pages 19–24, Hamilton, New Zealand, July 11-12 2002.

[Med01] Microsoft Windows Media. Understanding Secure Audio Path. Technical report, Microsoft, 2001.

[Mic07] Microsoft. Zune, 2007. http://www.zune.net/en-US/press, accessed on 7th of Jan.2007.

[MUNS05] Fabian Mörchen, Alfred Ultsch, Mario Nöcker, and Christian Stamm. Visual mining in music collections. In *Proceedings of the 29th Annual Conference of the German Classification Society (GfKl)*, Magdeburg, Germany, 2005. `http://www.mathematik.uni-marburg.de/~databionics/de/ /downloads/papers/moerchen05visual.pdf`.

[NDR05] Robert Neumayer, Michael Dittenbach, and Andreas Rauber. PlaySOM and PocketSOMPlayer: Alternative Interfaces to Large Music Collections. In *Proceedings of the Sixth International Conference on Music Information Retrieval (ISMIR)*, pages 618–623, London, UK, September 11-15 2005.

[Nok] Nokia. Webpage. http://www.nokia.de/, accessed on 9th of Jan.2007.

[Nok07] Nokia. Mobile Media API Support in Nokia Devices. Technical report, April 2007.

[Pam01] Elias Pampalk. Islands of Music: Analysis, Organization, and Visualization of Music Archives. Master's thesis, Vienna University of Technology (TU Vienna), 2001.

[PRM02] Elias Pampalk, Andreas Rauber, and Dieter Merkl. Using smoothed data histograms for cluster visualization in self-organizing maps. In *Proceedings of the International Conference on Artificial Neural Networks (ICANN)*, pages 871–876, Madrid, Spain, August 27-30 2002.

[RF01] Andreas Rauber and Markus Frühwirth. Automatically analyzing and organizing music archives. In *Proceedings of the 5. European Conference on Research and Advanced Technology for Digital Libraries (ECDL)*, Springer Lecture Notes in Computer Science, Darmstadt, Germany, Sept. 4-8 2001. Springer.

[RMD02] Andreas Rauber, Dieter Merkl, and Michael Dittenbach. The Growing Hierarchical Self-Organizing Map: Exploratory Analysis of High-Dimensional Data. *IEEE Transactions on Neural Networks*, 13(6):1331–1341, November 2002.

[RPM02] Andreas Rauber, Elias Pampalk, and Dieter Merkl. Using Psycho-Acoustic Models and Self-Organizing Maps to create a Hierarchical Structuring of Music by Musical Styles. In *Proceedings of the 3rd International Symposium on Music Information*

Retrieval (MUSIC IR), pages 71–80, Paris, France, October 13-17 2002.

[RPM03] Andreas Rauber, Elias Pampalk, and Dieter Merkl. *The SOM-enhanced JukeBox: Organization and Visualization of Music Collections based on Perceptual Models*, volume 32, pages 193–210. June 2003.

[Rub91] Dean Rubine. Specifying Gestures by Example. In *Proceedings of the 18th annual conference on Computer graphics and interactive techniques (SIGGRAPH)*, pages 329–337, New York, NY, USA, 1991. ACM Press.

[Sch06] Markus Schedl. The CoMIRVA Toolkit for Visualizing Music-Related Data. Technical report, Department of Computational Perception, Johannes Kepler University Linz, June 2006.

[Tza02] G. Tzanetakis. *Manipulation, Analysis and Retrieval Systems for Audio Signals*. PhD thesis, Princeton University, 2002.

[WIK05] WIKI. Mobile phones, 2005. http://en.wikipedia.org/wiki/Mobile_phone, accessed on 6th of Jan.2007.