JOHANNES KEPLER
UNIVERSITÄT LINZ
Netzwerk für Forschung, Lehre und Praxis

Mobile Service Oriented Architecture in the Context of Information Retrieval

MASTERARBEIT

zur Erlangung des akademischen Grades

Diplom-Ingenieur

im Masterstudium

INFORMATIK

Eingereicht von:
Christian P. Praher Bakk.techn., 0255763

Angefertigt am:
Institut für Informationsverarbeitung und Mikroprozessortechnik (FIM)

Betreuung:
o.Univ.-Prof. Dr. Jörg R. Mühlbacher

Linz, Juni 2008

Kurzfassung

Diese Arbeit beschäftigt sich mit Mobile Computing im Allgemeinen und im Umfeld einer Enterprise Search Infrastruktur im Speziellen. Der weitläufige Begriff des Mobile Computing ist dabei besonders auf so genannte „Smartphones“ bezogen, welche die Eigenschaften von Mobiltelefon, persönlichen digitalen Assistenten (PDA) und elektronischen Frezeitgeräten, wie etwa Kamera oder MP3-Player, in einem Gerät vereinen.

Ziel der Arbeit ist es sowohl einen allgemeinen Überblick über das Feld des Mobile Computing und dessen Möglichkeiten der Applikationsentwicklung zu bieten, als auch anhand zweier Prototypen eines mobilen Suchclients für eine Unternehmenssuche, konkrete Implementierungen mobiler Anwendungen zu zeigen.

Im ersten Teil wird deshalb nach einem kurzen Überblick über das Thema zunächst auf die wichtigsten Betriebssysteme und Entwicklungsplattformen im Bereich der Smartphones eingegangen. Weiters wird die mobile Webapplikationsentwicklung von ihren Anfängen bis heute beschrieben. Ein wesentlicher Aspekt der Arbeit liegt auf dem Thema der Service Orientierten Architekturen (SOA), speziell in einer ihrer üblichsten Realisierungen in Form von Web Services. Nach einer kurzen allgemeinen Einführung in das Gebiet der Web Services, werden speziell deren Eigenheiten im Hinblick auf mobile Anwendungen aufgezeigt.

Der zweite Teil beschäftigt sich dann mit dem im Titel beschriebenen Thema des *Information Retrieval*. Anhand zweier Prototypen, eine AJAX basierte für das *iPhone* optimierte Webanwendung, sowie ein nativer Client auf Basis des neuen Smartphone Betriebssystems *Android*, werden die unterschiedlichen Möglichkeiten des Zugriffs auf die Geräte-Daten näher beschrieben. Beiden Anwendungen gemeinsam ist dabei die Kommunikation mit dem Such-Server über ein HTTP basiertes REST Web Service.

Abstract

This thesis is about mobile computing in general and in the context of enterprise search and information retrieval in particular. Within this paper, the term of mobile computing is generally referred to "smartphones". A smartphone represents the convergence of cell phones, Personal Digital Assistants (PDA) and consumer electronic devices, such as cameras or mp3-players, into one single device.

The paper aims at pursuing two goals which are, to provide a general overview of the field of mobile computing and the different ways of creating mobile applications, as well as showing the concrete implementation of mobile applications on the basis of two search client prototypes for an enterprise search scenario.

The first theoretical part is concerned with todays most important smartphone operating systems and development platforms. Furthermore mobile web application development from their commencements until today will be described in detail. A key aspect of this part of the paper is to provide an overview of Service Oriented Architectures (SOA) with a particular focus on their realization by means of Web Services. After a brief general introduction of Web Services, their specialties with respect to a mobile environment will be highlighted.

The second part of the paper deals with the issue of information retrieval. In particular the various possibilities of accessing data stored locally on the device will be examined by means of two different prototype applications. The first client represents an AJAX driven web application tailor-made for Apple's *iPhone*. The second prototype shows the implementation of a native client on basis of the new smartphone operating system *Android*. Both applications in common is that they communicate with the enterprise search server via an HTTP based RESTful Web Service interface.

Danksagung

An dieser Stelle möchte ich mich recht herzlich bei all den Leuten bedanken, die ganz wesentlich zum Gelingen dieser Arbeit beigetragen haben.

Zunächst gilt mein Dank o. Univ.-Prof Dr. Jörg R. Mühlbacher der mir ermöglicht hat diese Diplomarbeit am Institut für Informationsverarbeitung und Mikroprozessortechnik (FIM) zu verfassen und mich während des gesamten Projektes stets vorbildlich betreute.

Besonderer Dank gebührt den Mitarbeitern der Firma Mindbreeze Software GmbH, allen voran DI Daniel Fallmann und DI Jakob Praher, die das Projekt stets mit höchster Priorität behandelten und mich immer voll unterstützt haben und die ich jederzeit mit meinen Anliegen behelligen konnte. In vielen gemeinsamen Treffen konnten wir das Projekt Idee um Idee verfeinern und ich durfte einen spannenden Einblick in die Arbeitsweise einer jungen, dynamischen IT-Firma gewinnen.

Weiters möchte ich mich recht herzlich bei meiner Familie und meiner geliebten Freundin Ariadne Köppl bedanken, die mich stets moralisch unterstützt haben und mir immer volles Verständnis entgegengebracht haben, auch wenn ich in den vergangenen Monaten nur sehr wenig Zeit für sie entbehren konnte.

Contents

List of Figures

List of Tables

Listings

Chapter 1

Introduction

1.1 Problem Formulation & Motivation

This thesis is motivated in the need of the Linz based software development company Mindbreeze Software GmbH[1], to develop a mobile client for their product "Mindbreeze Enterprise Search (MES) 3.0". Mindbreeze is a young software business focussing on the development of market-leading search technologies with concrete products in the sectors of enterprise, desktop and website search.

The mobile computing market is currently rapidly evolving with high growth prospects and almost daily announcements of new devices and application platforms, which results in an increasing diversification of devices, operating system and development platforms. Compared to more traditional information technology markets like the one of desktop computing, mobile computing is much less consolidated and neither standards nor even industry standards have yet been established.

Against this background, the first objective of the thesis is to identify and summarize which mobile application development platforms exist today and what capabilities each of them offers. If possible a candidate is to be found that serves as a common development platform for multiple operating systems. Another issue inseparably connected to the question of the underlying operating system is what data it reveals and how this data can be used to support the "user on the go". As opposed to desktop operating systems, mobile operating systems directly manage the applications that typically characterize a mobile device like e.g. Personal Information Manager (PIM) (address book, calendar dates, notes, task lists, ...), call logs, SMS/MMS or GPS positioning data, etc. As a consequence the degree of access to this information varies from system to system. This information is especially

[1] http://www.mindbreeze.com, last viewed 2008-04-23

relevant for an intelligent mobile client that not just considers the data provided by the server but also that stored on the local device.

Due to the enormous heterogeneity of today's mobile devices, no assumption has been made in advance of how exactly a mobile enterprise search client has to look like. The *what* and *how* have been basically left subject to the findings drawn from the research accompanying the thesis. Besides *native applications*, *mobile web applications* have been considered a possible platform for the search client. Hence a close examination of mobile web applications and how they can be developed will also be presented in this thesis.

Another key aspect that was clear from the beginning was that any developed prototype should follow the paradigm of a Service Oriented Architecture (SOA). SOA basically means that two applications communicate over a well defined and uniform interface. This allows for a decoupling of the two applications and reduces the mutual dependencies to a necessary minimum. In the light of the ongoing fast evolution of the mobile computing market this is an indispensible claim as any tight bonding to the server application should best be avoided. Instead the already defined Mindbreeze SOA interfaces should be used as the means of communication.

1.2 Outline of the Thesis

Basically the thesis is divided into two main parts. A theoretical, analytical part about the most important concepts and technologies of mobile computing that spans chapters 2 to 5. And a practical part encompasing chapter 6 that describes the prototypes that have evolved from this paper.

At the beginning chapter 2 serves as a brief introduction into the field of mobile computing and prepares the terminology used throughout the subsequent chapters of the thesis.

Chapter 3 provides an overview of the status quo of today's most common smartphone operating and development systems. Alongside the most important native platforms, also Java ME as a possible cross operating system application (development) platform is discussed thoroughly.

The evolution of mobile web applications that come from a totally different background than their desktop counterparts but now more and more converge into the same direction, is described in detail in chapter 4.

The concepts of Service Oriented Architecture (SOA) and how it can be applied to mobile computing is presented in chapter 5.

The two prototypes developed alongside this thesis are described in chapter 6. The first one is a web client tailored for the iPhone, which shows the capabilities of modern mobile web applications. The second one is a native

application developed with Open Handset Alliance's new mobile platform Android, which highlights the benefits of a native client.

Finally, chapter 7 presents a brief summary of the most important findings of the paper.

Chapter 2

Definition of Mobile Computing

Mobile computing has become a real catchphrase over the past few years and the interpretations of the meaning of the term vary greatly. As a consequence, the possible definitions referred to by mobile computing span from portable laptop computers, over handheld devices to ubiquitous agent systems.

2.1 Smartphone – The Universal Mobile Terminal

Within the scope of this thesis, mobile computing is generally referred to *smartphones*, which represent intelligent phone centric handheld devices that can be leveraged by third-party applications [87]. Smartphones are believed by many to be the enabling device for mobile computing, similar to what the IBM PC was for desktop and office computing. They are sometimes dubbed the *universal mobile terminal* as they unify many functionalities that have as yet only been provided by individual devices [87]:

- **Communication**
  The features found in a standard mobile phone are of course also key components of every smartphone. The primary communication network usually is a cellular network like Global System for Mobile Communications (GSM) that offers the basic telephony functionality as well as Short Messaging Service (SMS) and Multimedia Messaging Service (MMS).

  Besides the traditional cellular network, smartphones usually also provide a high speed wireless (local area) packet switched network connection, most commonly in the form of Wireless LAN (WLAN).

- **Computing**
  A smartphone possesses an advanced operating system that allows for

leveraging the device with third-party applications. This sort of functionality was usually found in *Personal Digital Assistants (PDA)*. A PDA is a small mobile device that is specialized for Personal Information Management (PIM) applications like addressbook, calendar, task-lists, etc. The best known representative for a PDA is probably the *Palm Pilot* [87]. The average PDA did not have a telephone unit for voice communication.

- **Consumer Electronics**
  The average smartphone also encompasses consumer electronics functionalities with at least a built-in camera and a mp3-player. Together with the advent of new more powerful devices, more and more consumer electronics components are being integrated into smartphones. Amongst the currently most popular ones are GPS (Global Positioning System) receivers and accelerometers.

  Equipped with a GPS receiver a device can precisely[1] determine its position through the signals sent from the GPS satellites. This information can then be used by various applications e.g. route planners or map applications. Usually the devices also expose the GPS data via an API to third-party applications.

  An accelerometer is an electromechanical device that measures acceleration forces[2]. The measured forces are both static caused by the constant force of acceleration due to gravity, as well as dynamic caused by moving and vibrating the accelerometer. The data provided by the accelerometer can be e.g. used to determine the position of the device and automatically switch the orientation (portrait versus landscape), to quiet down the speaker when the device is dragged away from the ear while issuing a call, or for interactive games. Like GPS, accelerometer data is usually accessible via a native API.

A key aspect of smartphones is that they possess a complex operating system. This is a major difference that sets them apart from the class of *feature phones* [22]. A feature phone is a regular mobile phone with feature support such as e.g. a high-resolution display, a built-in camera or a mp3-player. Feature phones are closed devices that do not offer the extensibility through a native programming API. The only way of customization sometimes offered by these devices is through Java ME or via web applications. As will be shown in section 3.2 the level of Java support usually is very restricted and far from uniform for different devices. Also web application support is usually very restricted due to the limited built-in browsers (see chapter 4).

[1] GPS singals usually provide an positioning accuracy as low as several meters. See http://www.gartrip.de/long.htm, last viewed 2008-04-23

[2] http://www.dimensionengineering.com/accelerometers.htm, last viewed 2008-04-23

As figure 2.1[3] shows, smartphones take only a small slice of the overall handset market compared to feature phones. This is mostly motivated in the higher prices of smartphones and the different billing models. Usually feature phones are bundled for free with a new mobile contract, whereas smartphones have to be bought separately. Still smartphones already play a very important role for business customers and are expected to become increasingly important for non-corporate customers due to the ongoing convergence with consumer electronic devices. According to figure 2.1 smartphones will be the type of handset device that will experience the biggest relative growth in terms of percentages.

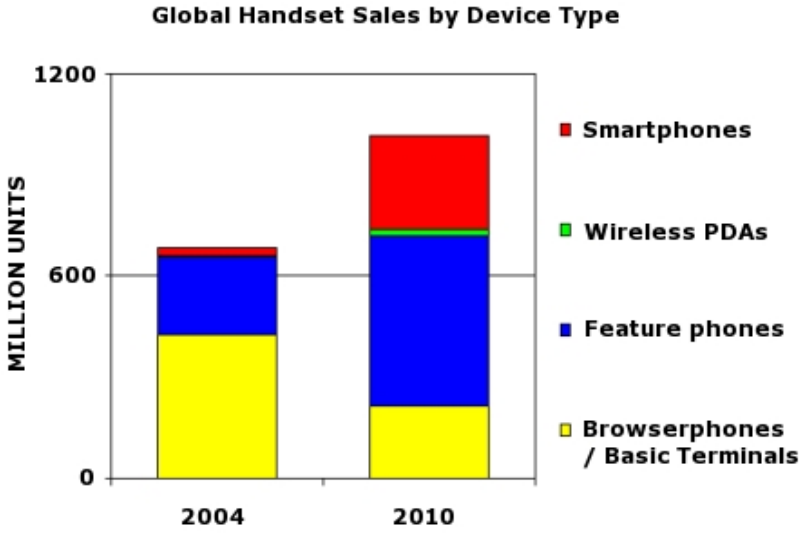


Figure 2.1: Global handset sales by device type

Due to the heterogenous heritage of smartphones, their form factor may also vary greatly. Possible are devices with a full QWERTY[4] keyboard as well as devices that only offer a touch screen and have no keys at all, or combinations of these two variants.

Throughout the paper, the terms *smartphone*, *mobile phone*, *mobile device* and *handset* are used interchangeably.

[3] http://www.linuxdevices.com/news/NS2742315828.html, last viewed 2008-04-24

[4] QWERTY is a short form for describing a complete keyboard as present on a desktop computer. The word derives from the first six characters found in the first character row of an English keyboard: Q, W, E, R, T, Y

2.2 Diversity of Networks

Smartphones usually possess many kinds of network connections ranging from wide range cellular networks like GSM or UMTS, over medium range packet-switched networks like Wireless LAN, to low range networks like Bluetooth or Infrared (IrDa). The following is an overview of today's most important network types for smartphones. This is not a complete or technical substantiated list, but should rather provide an overview about common transfer rates and the resultant possibilities for network-centric mobile applications. A thorough analysis of current and future mobile networks can be found in [87].

2.2.1 Cellular Networks (MAN/WAN)

Cellular networks are the most important type of network for smartphones. Since their inception in the 1960s, they have advanced a lot and transfer rates have improved significantly [87]:

First Generation (1G) The first analog cellular networks emerged in the 1960s but never received much attention due to their low capacities. In 1982 the networks were improved and offered already as much as 832 channels and much higher capacities than the early versions. But due to incompabilities between different first generation systems and a general lack of standardization, these networks never achieved mass market attention.

Second Generation (2G) The second generation cellular networks were the first systems that attracted overwhelming market interest. Synonymous for this generation are the European *Global System for Mobile (GSM)* as well as the North American *Code-Division Multiple Access (CDMA)*. GSM was originally proposed by the European nations to design a pan-European mobile communication network, whereas CDMA was primarily designed and fostered by the Californian company Qualcomm[5]. As CDMA is primarly employed in North America, the further focus will be on GSM and its successor technologies.

The digital and circuit-switched GSM operates on the 900-MHz and 1800-MHz frequency bands in Europe and Asia. Due to the Time-Division Multiple Access (TDMA) incorporated in GSM, multiple users can communicate over the same frequency simultaneously with every user receiving a very short interval in its own fix time slot. This allows eight users to share a single 200-KHz channel in time-divison manner. Together with GSM the first rudimentary cellular network data-centric application – *Short Message Service (SMS)* – was introduced. SMS allowed for 160 characters to be sent over

[5] http://www.qualcomm.com/, last viewed 2008-04-23

the GSM network and was a huge success in Europe and Asia. Still GSM suffered from very low data rates of only 9.6 to 14.4 Kbps which was mainly because of the circuit-switched nature of GSM. Another disadvantage of data transfer over the circuit switched GSM was that the channel for transmission (one of the eight time slots in TDMA) was always blocked, even if no data was sent. This is no problem for voice communication where a solid steady network connection is needed as long as both parties communicate. But it is problematic for data transmission that is usually characterized through bursty peeks and periods with no transfer at all.

2.5G Systems As the need for data services such as E-Mail, Multimedia Messages (MMS) or web browsing constantly increased, the aim was to leverage the widely deployed GSM network with a packet-switched system for value added data services. As a consequence the so called 2.5 G systems were predominantly enhancements for data transfer on top of the existing GSM network. The first exponent of this class of systems was *General Packet Radio Service (GPRS)* that first went into operation in 1999. Theoretically GPRS offers a data rate of up to 72.4 Kbps, but this can only be achieved if all eight time slots are used aggregately. Since GPRS is designed to coexist with traditional GSM voice services, in reality the achievable data rate is considerably lower. A remarkable advantage of GPRS over direct data transfer via traditional GSM is that, since no channel has to be reserved, a connection seems always on as long as the device remains active in a GSM network.

A further step toward 3G high speed cellular networks was the introduction of *Enhanced Data Rates for Global Evolution / Enhanced Data Rates for GSM Evolution (EDGE)*. EDGE uses a different modulation scheme than GSM/GPRS which allows for a much higher bit rate. If all eight available time slots are used the data rate can be as high as 384 Kbps. But as with GPRS this high data rates are usually not achieved in reality. Nevertheless, due to its comparatively high data rates, EDGE is often classified as 2.75 G system.

Third Generation (3G) 3G systems currently mark the top in the evolution of cellular networks. The most important manifestation of 3 G in Europe is the *Universal Mobile Telecommunications System (UMTS)*. UMTS represents the direct predecessor of GSM/GPRS/EDGE. UMTS can be considered an incremental technology as the specification of UMTS is done in phases[6]. Each phase in the UMTS evolution is called a "release". The first release was named *release 99* after the year of its publication. The subsequent releases were just given sequence numbers. Release 99 was built on GSM and

[6] http://www.telecomabc.com/u/umts-releases.html, last viewed 2008-04-23

featured full backward compatibility with GSM as well as interoperability between GSM and UMTS. The data rates were 64 Kbps circuited-switched and 384 Kbps packet switched (This results from EDGE being the direct predecessor of UMTS). Another notable evolution was release 5 which introduced *High Speed Downlink Packet Access (HSDPA)*. With downlink data rates of up to 10 Mbps HSDPA is considered a key enabler technology for rich Internet applications on mobile devices. The counterpart of HSDPA, *High Speed Uplink Packet Access (HSUPA)*, was introduced in release 6 and offers increased up-link speeds of more than 5 Mbps. Eventually the *UMTS Long Term Evolution (LTE)* will lead the system to a full-fledged fourth generation network with WLAN integration, IPv6 support and data rates of up to 100 Mbps[7].

Fourth Generation (4G) Fourth generation systems will be exclusively packet-switched and are therefore referred to as the *all IP networks*. Wired and wireless services are expected to converge and the resulting networks will provide high data rates of 20 to 100 and even 1000 Mbps. This will eventually bring the full multimedia experience to mobile devices with video streaming and realtime audio support. It will also allow for advanced client-server applications with frequent interaction and data exchanges of large files. As a consequence of an all IP network, voice will be transferred over the same packet-switched infrastructure as data, in the form of Voice over IP (VoIP).

2.2.2 Wireless LAN (802.11)

Wireless Local Area Networks (WLAN) fall into a different field of application than cellular networks. Firstly, they operate in much more limited ranges between several meters and a few tenths of a meter. Secondly, they are always packet-switched instead of circuit-switched and thus primarily data-centric. They also deliver much higher transfer speeds than the average cellular network, with data rates of usually 11 to 54 Mbps. This makes them the ideal complement to the voice-centric cellular networks for value added data-services like e.g. web browsing, E-Mail or multimedia consumption. Often 802.11 wireless LANs are referred to as *Wi-Fi*. Wi-Fi is a certification of the Wi-Fi Alliance[8] that tests and certifies products for compliance with the various 802.11 IEEE specifications.

[7] http://www.funkschau.de/heftarchiv/pdf/2007/fs_0707/fs_0707_s34-s35\%20UMTS-LTE.pdf (german), last viewed 2008-04-23

[8] http://www.wi-fi.org, last viewed 2008-04-24

The two currently most common types of 802.11 Wireless LAN are [87]:

- **802.11b**
  The 802.11b wireless LAN specification was released in 1999. It operates on the unlicensed 2.4-GHz frequency band (This is the same frequency as the one used by microwave ovens). Because the 2.4-GHz spectrum is free, anyone can operate a 802.11b wireless LAN router without permission from the government or having to pay licence fees. The maximum offered data rate is 11 Mbps and the maximum range of operation is up to 100 meters. The value of 100 meters is rather theoretical and can normally only be achieved outdoors, if only very few obstacles (e.g. walls) are between the sender and the receiver.

- **802.11g**
  802.11g was introduced in 2003 and operates on the same 2.4-GHz frequency band as 802.11b. Compared to 802.11b, 802.11g wireless LAN offers a significantly higher data rate of up to 54 Mbps. Most modern smartphones are both 802.11b and 802.11g compatible.

2.2.3 BAN/PAN

Body Area Networks (BAN), or *Personal Area Networks* are the third type of networks usually found in a smartphone. As their name implies, they operate on very low ranges in the immediate vicinity of the user. They are commonly used as wireles *cable replacement* (e.g. connecting an headset to a mobile or for data synchronization with a desktop PC) and for *ad hoc* data networking (e.g. mobile to mobile data transfer). The offered data rates are rather low but usually sufficient for the described functionalities. Bluetooth and Infrared are the most important *Wireless Personal Area Networks (WPAN)* for mobile phones [87]:

- **Bluetooth**
  Bluetooth was invented in 1994 by phone manufacturer Ericsson. It is characterized by a short operation range of 10 to 100 meters and a low data rate of 1 to 2 Mbps. Its low power consumption of only 10 to 100 mW and the low hardware unit price (less than $5) make it ideal for small wireless devices like mobile phones. It operates on the same unlicensed 2.4-GHz frequency band as 802.11 b/g.

- **Infrared (IrDA)**
  The Infrared Data Association (IrDA)[9] is a nonprofit organization that develops globally adopted specifications for infrared wireless communication. Although infrared is used in similar scenarios as Bluetooth, the underlying techniques are quite different. While Bluetooth uses

[9] http://www.irda.org/, last viewed 2008-04-24

radio waves for data transfer, infrared uses beams of invisible light to transmit information[10]. By contrast to Bluetooth, infrared requires line-of-sight (los) for communication. Due to the average data rates of only some several hundert Kbps and the much more limited operation range of about 1 meter[11] as well as the necessary line-of-sight, infrared has been generally superseded by Bluetooth within the newer smartphone devices.

2.3 Key Limitations & Application Development Challenges

Mobile computing applications are bound to some specific limitations, which their desktop counterparts are usually not faced with or at least not to such high extent. Amongst others the most important ones are:

- **Battery Life**
  Power consumption and limited battery life is one of the biggest problems mobile devices are facing today. The bottleneck of battery life time is even considered to be so critical that it could seriously challenge Moore's Law [12, 25, 32, 59]. While engineers actually manage to double the number of transistors on an integrated circuit every 18 months or at least every second year, power consumption could denote a physical barrier.

  Besides the processor, mobile devices also include many other components that show a significant power consumption, e.g. the display, network devices or the graphics card. Figure 2.2 [59] shows a comparison of the peak power dissipation of 2G and 3G cellular network technologies.

  From an application developers perspective this means that heavyweight calculations and frequent network transmissions should be avoided whenever possible.

- **Processer Performance**
  The processor clock speed of modern mobile devices is significantly lower than that of desktop computers. One of today's most common smartphone processor architectures is Advanced RISC Machine (ARM), which operates (at the time of writing of this thesis) on a clock rate of considerably less than 1-GHz on a single core. By contrast average desktop computers have dual-core CPUs which are clocked at more than 2-GHz.

[10] http://www.phonescoop.com/glossary/term.php?fid=36, last viewed 2008-04-24

[11] http://www.gsmfavorites.com/documents/bluetooth/compared/, last viewed 2008-04-24

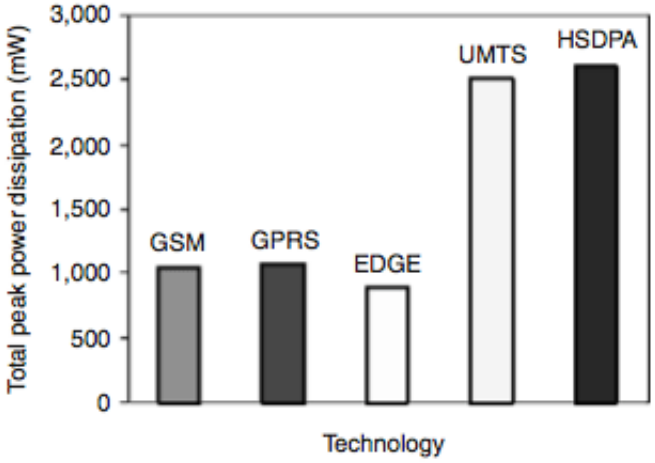


Figure 2.2: Power consumption of cellular technologies

- **Memory and Storage**
  Analogous to the processor performance, mobile devices are also limited in respect to main memory and permanent storage. Typical smartphones have no hard disk like desktop computers, but instead flash memory cards. While flash memory cards offer much less storage space compared to hard disks, they offer many other advantages crucial for mobile devices [87]:
  - Faster access times
  - Smaller and lighter than hard disks
  - No error-prone mechanical parts
  - Quietness
- **Form Factor**
  Whereas desktop computers always feature the more or less same appearance with a full-fledged keyboard, a mouse as input device and a relatively big screen, smartphones may come in very different shapes.

  The input device could be a standard cell phone 12-button keypad (numbers 0 to 9 as well as *, +, # keys and usually additionally some function keys for dialing), the already mentioned QWERTY keyboard, an alphabetic keyboard with keys arranged alphabetically as well as touch screens with stylus-based or finger-tip input.

  The display may also vary greatly according to the general form factor of the device. Apple's full touchscreen iPhone e.g. features a display

with a resolution of 320 x 480 px[12]. By contrast the BlackBerry 8800 with a QWERTY keyboard offers a resolution of only 320 x 240 px[13].

- **Changing Device Context**
  Desktop computers always operate in the same context. Once installed at a specific place, they usually remain there until the end of their lifetime. This is opposed to mobile devices which are operated in ever-changing scenarios. For a mobile application it may be important to know about this augmented context and react accordingly.

- **Personal Information Management (PIM)**
  PIM applications like address book, calendar, e-mail, etc. are also found on desktop computers, but for mobile devices they have a particular relevance. On desktop computers they are usually independent of the underlying operating system and installed as third-party applications. On mobile devices PIM functionalities including telephony and SMS/MMS are core features of the underlying operating system. The level of support for accessing PIM data varies from platform to platform.

- **Security & Privacy**
  Last but not least, mobile devices are subject to more security threats than desktop computers. One of the biggest being loss and theft [87, Chp. 6]. This is a threat mostly unknown and insignificant for wired desktop applications as they are usually operated behind (secure) corporate walls.

  Another serious endangerment comes from the heavy interconnection in the various previously described wireless networks that are by default more insecure than their wired companions.

  As described, the user of a mobile device also reveals more private data than the average desktop application user. Most notably this sensitive data comes from sensors like GPS, or from very personalized applications like e.g. contact lists, calendars or phone logs.

[12] http://www.apple.com/iphone/specs.html, last viewed 2008-04-24

[13] http://www.mobiletechreview.com/phones/BlackBerry_8800.htm, last viewed 2008-04-24

Chapter 3

Native Development Platforms

3.1 Overview

Alongside the myriad of mobile devices there exists also a huge number of operating systems and even more application development platforms.

This chapter aims at giving an overview of today's most important smartphone operating systems and how applications can be developed for them. Table 3.1 shows on overview of the smartphone operating systems market share sales of quarter 4 2007, researched by analyst house *Canalys* [11].

Operating System	**Market Share Q4**
Symbian OS	65 %
Windows Mobile	12 %
RIM Blackberry	11 %
iPhone OS	7 %
Linux (cumulative)	5 %
Palm/Garnet OS	0 %

Table 3.1: Worldwide smartphone market shares by operating system in Q4 2007

3.2 Java Platform, Micro Edition (Java ME)

Java Platform, Micro Edition (Java ME, formerly J2ME) differs from the other mobile platforms introduced in this chapter, in that it is no mobile operating system. Instead it is a *middle layer* between a specific mobile operating system and value added services and applications offered by a service provider [87]. The Java Platform is divided into three main products, each one targeted at a special field of application [71]:

- **Java Platform, Standard Edition (Java SE, formerly J2SE)**
  Java SE serves as the standard Java Edition for developing desktop and small server applications. It provides a rich set of APIs for all various kinds of software development needs, from basic String manipulation over graphics and user interface (UI) creation to networking. It is the basis for the Enterprise Edition (Java EE).

- **Java Platform, Enterprise Edition (Java EE, formerly J2EE)**
  Java EE extends the Standard Edition by adding a set of libraries for creating full featured web- & server-applications. It offers e.g. transaction support, libraries for mapping objects to relationale databases (O/R-mapping) and Web Services. It is specially known for its Servlet-Technology, which allows for using Java's rich libraries within web-applications.

- **Java Platform, Micro Edition (Java ME, formerly J2ME)**
  Java Micro Edition is specially aimed for the development of applications on small, limited devices like mobile phones, PDAs, set-top boxes and so forth. It utilizes only a subset of the APIs available with the Standard Edition and builds on a smaller Virtual Machine (VM) for meeting the resource scarcity of its target devices.

All editions in common is the basic architecture of the Java platform. The term *Java* does not only refer do a particular language-syntax, it much more represents an entiry platform, comprising the following three fundaments [71]:

- **The Java programming language**
  The Java programming language is a high level object oriented language that is syntactically smimilar to its ancestors of the C-family of programming languages (C/C++). It avoids the use of unsafe pointers and offers the programmer automatic memory management with garbage collection of unreferenced objects. Another cornerstone of the Java programming language is its inherently object oriented nature.

- **The Java virtual machine**
  The concept of a virtual machine (VM) is the foundation of the Java platform. It provides a layer between the native operating system and the applications written in Java. In that way Java applications are platform independend and can be run on any platform (having a Java VM) without the need of recompiliation. Sun Microsystems, the founder of Java, therefore coined the term "write once run anywhere". As will be shown later in this chapter this metaphor does not entirely hold true for mobile computing.

- **The Java Application Programming Interfaces (APIs)**
  The rich set of standard libraries is one of the main reasons of the success of the Java platform. The spectrum of the functionalities provided by these libraries range from basic String manipulation, I/O[1], GUI[2] to concurrent and network programming.

Given this nature and the fact that Java was originally designed as a platform for mobile embedded devices, like e.g. set-top boxes [10], it should be ideally suited for the development of mobile applications. Java has proven to be one of the most successful development environments for mobile computing. According to Sun Microsystems Java has already been deployed on more than 1.2 billion handsets by June 2006 and 8 out of 10 newly shipped phones today are Java ME technology enabled [69]. This makes Java ME at present the most ubiquitous application development platform for mobile devices.

In light of the fact that Java is not only available on sophisticated smartphones running a fancy operating system but also on much cheaper feature phones, it sounds like an extremly promissing platform for mobile software development.

But Java ME can not be seen as an homegenous platform that is the same on any Java enabled device. Instead it has evolved and frequently changed along with the devices its being deployed to since its introduction in the year 2000 [68], leaving the entire platform quite fragmented. The following is an overview of the basic architecture and the most notably steps taken to consolidate the platform, resulting in the different flavours of Java ME that are now available on handsets.

3.2.1 Basic architecture & GEN-1

To best fit the different needs of the various mobile devices, the Java ME architecture is divided into different components (figure 3.1 [62]).

The basic blocks of the platform are *configurations*, *profiles* and *additional APIs*, which build on each other in a layered fashion:

- **Configurations**
  A configuration is the most basic building block, dealing with VM specifications and offering the most basic APIs. It is tailored for the needs of a specific group of devices, e.g. handsets with less than 512KB of memory and a CPU speed of 50 to 200MHz. The VM is either a full Java Virtual Machine (JVM) or a subset, like the Kilobyte Virtual Machine (KVM). The currently offered configurations are the Connected Limited Device Configuration (CLDC) for small resource constrained devices, as well as the Connected Device Configuration (CDC)

[1] Input/Output
[2] Graphical User Interface

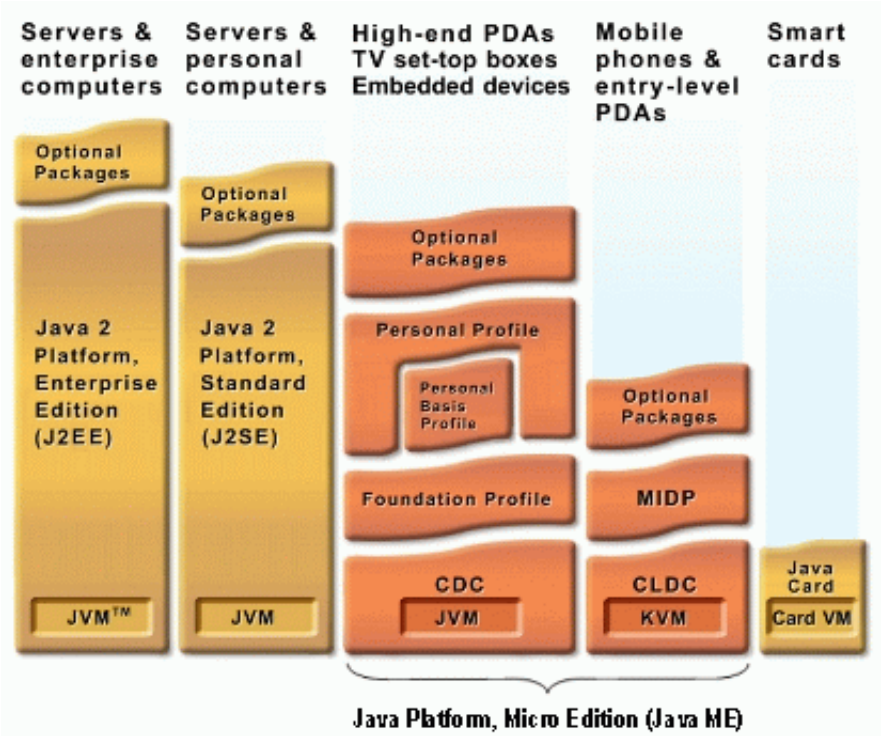


Figure 3.1: Overview of Java ME within the Java family of technologies

for wireless devices with greater computing power. Presently almost all Java-enabled mobile devices build on the CLDC.

- **Profiles**
  Profiles represent the next layer in the Java ME architecture, by directly building on a certain configuration. They add support for more specific APIs and thus create a closed application development framework. Typical APIs offered by a certain profile encompass application life cycle management, user interface (UI) and persistent storage. The most widespread profiles are Mobile Information Device Profile (MIDP) 1.0 and 2.0 which both build on the CLDC.

- **Optional APIs**
  The optional APIs depict the top layer of the Java ME architecture chart. Mostly these are libraries that are strongly related to mobile computing, but can not be included by default due to special requirements, like a certain piece of hardware or computing power. A typical representative of an optional package is the Bluetooth API (JSR 82). Although very closely related to mobile wireless computing, not every device can a priori be expected to have built-in bluetooth support.

Other characteristic packages include support for e.g. Web Services (JSR 172), 3D-Graphics (JSR 184) or the Session Initiation Protocol (SIP) (JSR 180).

Limited versus Connected Device Configuration

As shown in figure 3.1, Java ME is basically divided into two stacks of which one is based on the Connected Device Configuration (CDC) and the other one builds on the Connected Limited Device Configuration (CLDC). As described above, a configuration is the most basic building block in the Java ME framework. It defines the most fundamental properties of a particular Java ME enviroment like e.g. the virtual machine (VM) and the basic class libraries. As the Java ME depicts a layered architecture, necessarily all layers that are further up in the stack ultimately depend on the fundament and have to come up to the possibilities and boundaries defined in the lower levels.

The main difference between the CDC and the CLDC is that the CDC builds on a general VM whereas the CLDC rests upon a limited VM, the Kilobyte Virtual Machine (KVM) that has a specially low memory footprint and can thus be used on devices with little computing power. Such limited devices are e.g. ordinary handsets, which typically feature a 16- or 32-bit processor that runs at a clock speed of 50 to 200 MHz [68]. The overall minimum memory requirements are in the range of 160 kB to 512 kB (depending on the used profile) [64]. Currently the KVM is being replaced in favour of a minimalistic HotSpot VM that delivers the typical benefits of HotSpot-compilation to small resource-constrained devices [68]:

> *The CLDC HotSpot Implementation delivers nearly an order of magnitude better performance than the KVM while running in a similarly small memory footprint required by resource-constrained mobile phones and personal organizers. It delivers not only better performance, but also more robustness. The CLDC HotSpot Implementation is the recommended virtual machine technology for new product deployments in this class of devices ...*

The CDC by comparison may utilize the same JVM as Java SE applications, with a typical memory footprint between 1 MB and 10 MB [64, Figure 2-19].

As most of the Java enabled devices today build on the Connected Limited Device Configuration (CLDC), the focus of the evaluation of the Java platform within this paper will be on the CLDC. In fact support for the Connected Device Configuration (CDC) is rather limited and at present only four mobile devices[3] supported CDC out of the box[4].

[3] Nokia 9300/9500, SavaJE Jasper 20, SonyEricsson M600, SonyEricsson P990

[4] http://www.blueboard.com/javame/devices.htm, last viewed 2007-12-28

Also if a device offers support for CDC it typically is also capable of running CLDC applications. Therefore considering Java as a development platform for mobile applications today, it is inevitable to take a close look at the CLDC and the profiles building on it.

Basic configurations and profiles

The CLDC 1.0 (JSR 30) with the MIDP 1.0 (JSR 37) profile building on it were the basic working set of Java ME at its beginning in the year 2000. According to the specification [63] the following restrictions apply to CLDC 1.0 in contrast to Java SE:

- No floating point support
- No Java Native Interface (JNI)
- Limited error handling
- No user-defined class-loaders
- No support for reflection
- No object finalization
- No weak references
- No support for thread groups and daemon threads

The first official profile for the CLDC 1.0 was the MIDP 1.0 (JSR 37). It introduced some important features like the concept of MIDlets[5], GUI classes, persistent data storage and so forth.

More notably from today's perspective are probably the things CLDC 1.0/MIDP 1.0 did *not* support and which add as an exclusion criteria for developing mobile applications against this platform. MIDP 1.0 lacks e.g. support for certificates and as a consequence support for Secure Sockets Layer (SSL) or Transport Layer Security (TLS) [26]. Therefore secure connections over HTTPS can not be made with MIDP 1.0.

According to Mobref[6] MIDP 1.0 is currently deployed on approximately 11 % of the tracked handsets. Since MIDP 2.0 can also be deployed on basis

[5] A midlet is roughly the Java ME pendant to the Java SE Applet. It is usually refered to as MID Profile application. Any Java ME CDLC based application must extend the `javax.microedition.midlet.MIDlet` class to be able to run on the CLDC application stack.

[6] Mobref (http://www.mobref.com/statistics/, last viewed 2007-12-29) provides a comprehensive database of mobile devices and allows for querying for CLDC/MIDP or Symbian OS support amongst others. At the time of writing of this paper 1179 distinct devices were listed in the Mobref database. The Mobref statistical data is collected from visitors to the GetJar website. GetJar offers downloads of mobile software and has approximately 100,000 mobile visitors every day.

of CLDC 1.0 the market share of CLDC 1.0 is higher than that of MIPD 1.0 and amounts, according to Mobref, to circa 30 % of all available handsets.

To make up for the insufficiencies of the initial versions of CLDC and MIDP SUN Microsystems introduced its predecessors CLDC 1.1 (JSR 139) and MIDP 2.0 (JSR 118) in the year 2003 and 2002 respectively. Since MIDP 2.0 was released prior to CLDC 1.1 it does not necessarily depend on it. In fact many devices build on the CLDC 1.0 configuration with MIDP 2.0 as the basic profile.

Whereas changes and additions to MIDP 2.0 over 1.0 were quite fundamental, CLDC 1.0 was generally considered balanced and version 1.1 introduced a few new functionalities. According to the specification the following main differences exist between versions 1.0 and 1.1 of the CLDC [66]:

- Floating point support has been added
  - Classes `java.lang.Float` and `java.lang.Double` have been added
- Weak references support has been introduced
- Thread objects have names like threads in Java SE
- Classes `java.util.Date`, `java.util.Calendar` and `java.util.TimeZone` have been made more Java SE compliant
- Error Handling has been improved
- Minimum required memory budget has been increased from 160 KB to 190 KB, mostly due to the floating point support

Extensions from MIDP 2.0 to version 1.0 have been more profound and added to a considerable fragmentation of the Java ME. MIDP 2.0 introduced a lot of new functionalities and enabled many new applications that have not been possible with version 1.0. Especially in the realm of mobile enterprise application development, MIDP 2.0 introduced some indispensable features like SSL/TLS support or the not user initiated sending of messages (push architecture).

According to the MIDP 2.0 specification [65], the most important changes comprise:

- Support for certificates
- Support for secure connections by at least one of the following specifications
  - HTTP over TLS (HTTPS)
  - SSL V3

  - WTLS
  - WAP TLS profile and tunneling

- Extended connectivity by including the interfaces:
  - `CommConnection` (Serial port connection)
  - `HttpsConnection`
  - `SecureConnection` (Secure socket stream connection interface)
  - `ServerSocketConnection` (Server socket stream connection interface)
  - `SocketConnection` (Socket stream connection interface)
  - `UDPDatagramConnection` (For User Datagram Protocol (UDP) based network connections)
  - `PushRegistry` (For porgrammatically invoking MIDlets)
- Over-The-Air (OTA) provisioning (Discovery and download/installation of MIDlets)
- Improved multimedia capabilities
- Game API

The Mobref statistic states that roughly 74 % of the observed handsets had MIDP 2.0 support. Even if the figure is not representative for the overall mobile market it clearly shows that MIDP version 2.0 is much more prevalent than MIDP 1.0 of which the market share on Mobref amounts to approximately 11 %. CLDC 1.0 is available on circa 29 % of the listed devices whereas CLDC 1.1 adds up to 56 %.

3.2.2 Java Technology for the Wireless Industry (JTWI)

The fragmentation of the Java ME application framework starts with the distinction of the Connected Device Configuration (CDC) and the Connected Limited Device Configuration (CLDC). Within the more prevalent stack of CLDC there exist currently two different versions of configurations (CLDC 1.0 & CLDC 1.1) as well as two quite different profile specifications MIDP 1.0 & MIDP 2.0.

In response to the quickly evolving mobile device hardware market, additional APIs have been introduced to allow developers to exploit the new handset features. These APIs have been amongst others Bluetooth (JSR 82), 3D Graphics (JSR 184), Wireless Messaging (JSR 205), Web Services (JSR 172), and so forth. This process furthermore lead to a fragmentation of the market of Java ME enabled mobile devices and made it particularly difficult for the developers to rely on a basic set of functionality.

To counter this trend, SUN Microsystems introduced a new specification named *Java Technology for the Wireless Industry (JTWI)* (JSR 185) that guarantees developers to find a basic Java ME environment consisting of configuration, profile plus additional APIs on all compliant devices.

In detail the JTWI specification defines [67]:

- **Configuration**
  CLDC 1.0 (JSR 30) is mandatory but may be superseded by CLDC 1.1 (JSR 139)

- **Profile**
  MIDP 2.0 (JSR 118) is mandatory

- **Additional APIs**
  Wireless Messaging API (WMA) (JSR 120) must be present in JTWI compliant devices.
  Mobile Media API (MMAPI) (JSR 135) is an optional API, which must only be present if the target device makes use of video playback and audio or video/image recording within Java applications.

  Further optional APIs can be included by handset manufacturers to let Java developers optimally exploit their devices, but the JTWI specification makes no assumptions that such packages do exist.

Figure 3.2 [67] depicts an overview of the JTWI (JSR 185) components within a mobile phone software stack.

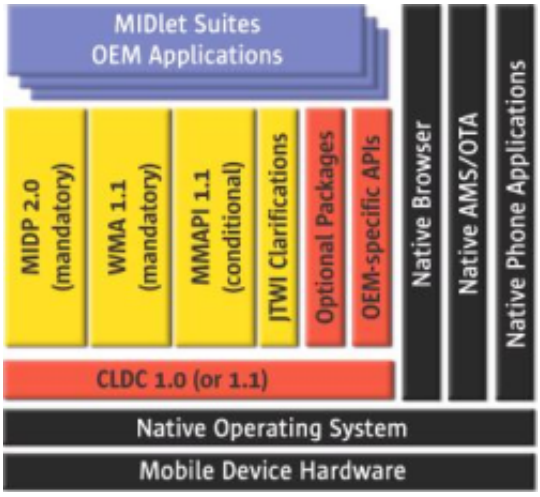


Figure 3.2: JTWI (JSR 185) components within the mobile phone software stack

3.2.3 Mobile Service Architecture (MSA)

As the wireless device market continues to evolve at a rapid pace, the JTWI specification needed a revision and several extensions to comply with today's mobile device market. The result is the Mobile Service Architecture (MSA) which is described in JSR 248 and was introduced in september 2006. It builds on many of the already existing Java ME specifications, most notably CLDC 1.1 (JSR 139), MIDP 2.1 (JSR 118) and the mandatory additional packages defined in JTWI (JSR 185).

As figure 3.3 from Sun Microsystem's official MSA-Website[7] shows, the MSA specification is divided into two platforms: MSA and MSA Subset. As the name suggests, MSA Subset is a true subset of MSA with fewer packages supported.

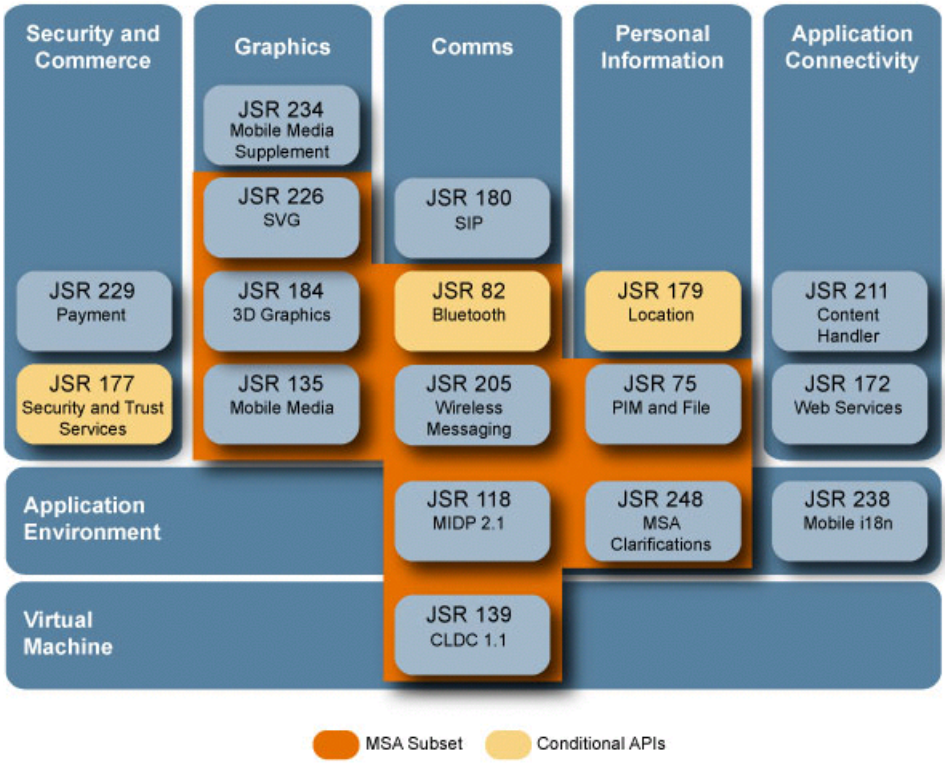


Figure 3.3: JSR 248 Mobile Service Architecture (MSA) chart

Due to their inherently hardware dependent nature, Bluetooth (JSR 82), Location (JSR 179) and Security and Trust Services (JSR 177) APIs are conditional even within the full MSA specification.

Because of its relative newness, there were only a handful of MSA com-

[7] http://java.sun.com/javame/technology/msa/, last viewed 2007-12-29

pliant devices on the market at the time of writing of the paper. According to SUN Microsystems official MSA device website[8], there are 12 compliant handsets available, 9 from Nokia and 3 from SonyEricsson. From this 12 devices only the 3 SonyEricsson models are full MSA compliant, whereas the 9 Nokia phones only offer MSA subset conformance.

3.3 Symbian - Symbian OS

Symbian OS is one of the most popular and widespread smartphone operating systems. Symbian Ltd. states in their presently latest sales and marketing brochure, *Market Round-Up* [72], that there have been shipped 20.4 million Symbian enabled smartphones in quarter 3 of 2007. Altogether more than 165 million Symbian OS powered handsets have already been sold. Currently Symbian runs on more than 134 different smartphone models from various manufacturers, including amongst others LG, Mitsubishi, Motorola, Nokia, SonyEricsson, Samsung, etc.

Symbian OS is especially prevalent in the European market with a market share of almost 90 % of all smartphone devices. It also plays significant roles in the Asian, African and South American regions, but it is not dominant in the North American market, where it falls far short of its competitors like Microsoft Windows Mobile, RIM BlackBerry, Garnet/Palm OS or Apple's Mac OS X iPhone, with only less than 5 % of the market.

Symbian OS originates from the EPOC operating system developed in the mid 1990s by Psion. EPOC was a 32-bit system programmed in C++ and was the successor of Psion's SIBO (SIxteen Bit Operating system). In 1996 the software team working on EPOC was formed into a new company called Psion Software, to allow for a better commercial licensing model, with the chance of licensing the EPOC OS to other Original Equipment Manufacturers (OEMs) than Psion [2]. The next step in the evolution of Symbian OS was the foundation of Symbian Ltd. in 1998 of which the original owners were Psion, Nokia and Ericcson. Today Symbian Ltd. is jointly owned by Nokia 47.9 %, Ericsson 15.6 %, SonyEricsson 13.1 %, Panasonic 10.5 %, Siemens 8.4 % and Samsung 4.5 %[9].

As a direct successor to EPOC of which the last release was 5, the first version of Symbian OS was 6.0. The most current release is Symbian OS version 9.5.

The Symbian OS is based on a microkernel design, meaning that only a small set of system function resides in the kernel. Originally this kernel was

[8] http://java.sun.com/javame/technology/msa/devices.jsp, last viewed 2007-12-29

[9] http://www.symbian.com/about/overview/ownership/ownership.html, last viewed 2008-01-30

not a real-time kernel but since version 9 Symbian OS is a real-time operating system [23]. Real-time systems have rigid time constraints and a system failure occurs if they are not met by an application. As the minimalistic microkernel of Symbian OS only provides the most basic functionalities like memory management, device drivers or power management, much of the remaining functions usually found in a monolithic kernel is provided by *servers*. By contrast to the kernel itself which operates in kernel-mode servers run in user-mode which reduces the potential for memory violations or similar fatal infringements. Servers are responsible for all kinds of low level tasks like socket connections, file handling, telephony, etc. *Clients* are the final building block in the basic Symbian OS architecture. They are "normal" applications that involve some kind of user interaction via the User Interface (UI). Together client, server and the kernel form a client/server architecture in which clients communicate with the servers via a message passing protocol (inter-process communication) and servers making executive calls into the kernel when necessary.

3.3.1 User Interface (UI) Platforms

Historicaly motivated and due to the fact that handheld devices come in many different form factors with diverse capabilities, Symbian OS does not provide a uniform User Interface (UI) platform. According to the official Symbian operating system guide[10], Symbian OS itself is "headless" , offering only the core frameworks and services and leaving it to the phone manufacturer to provide the right UI that both fits their device and market needs. Figure 3.4[11] depicts the architecture of Symbian OS. The top layer represents the proprietary UI platforms which are not part of the Symbian OS. Currently there are three UI platforms available on top of Symbian OS:

- **Nokia S60**
  Nokia S60 was formerly known as Series 60. Nokia also used to have other Symbian based UI platforms like the Series 80, which are now all being converged into S60. In its current release (3rd edition) it does not have touch screen support. Instead the devices come with support for various keypads like full QWERTY-keyboards, which makes them espescially suited for easy one-handed use.

- **SonyEricsson UIQ**
  UIQ has been originally owned by Symbian Ltd. SonyEricsson acquired UIQ in February 2007 and was joined by Motorola in October

[10] http://www.symbian.com/developer/techlib/v9.3docs/doc_source/guide/index.html, last viewed 2008-01-30

[11] http://www.symbian.com/developer/techlib/v9.3docs/doc_source/NewStarter/1-basics.html, last viewed 2008-01-31

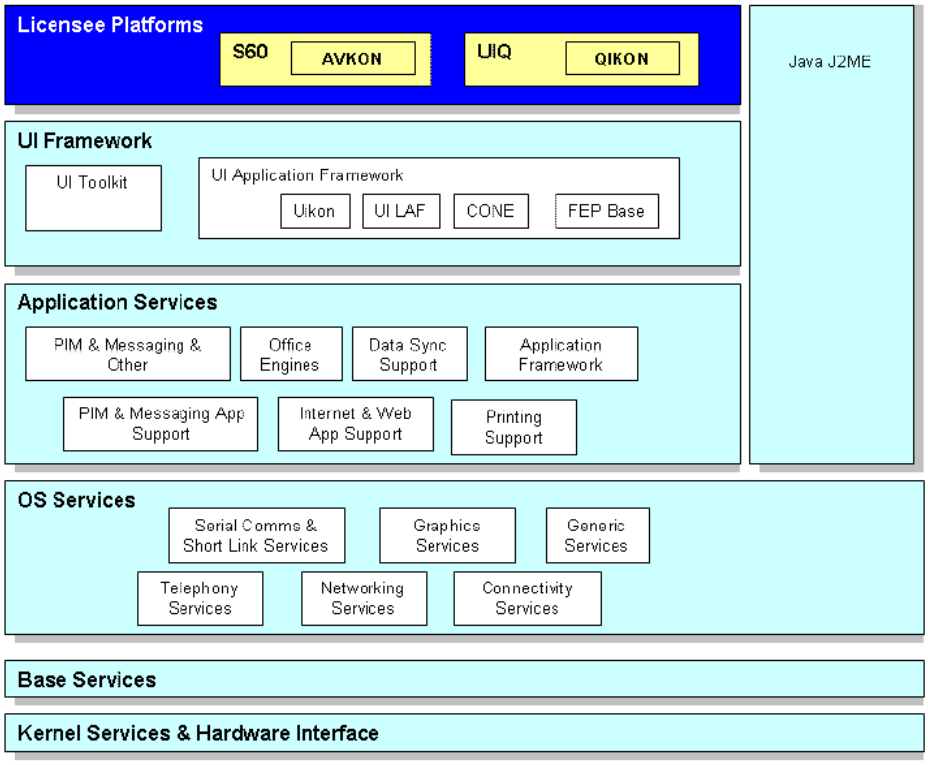


Figure 3.4: Symbian OS architecture chart

in ownership of UIQ [12]. It supports both touchscreen and keypad input and offers support for various screen resolutions.

- **FOMA MOAP**
  MOAP stands for Mobile Oriented Application Platform and is the Symbian UI of choice for the largest Japanese mobile operator NTT DoCoMo. FOMA is short for Freedom of Mobile Multimedia Access and is the brand name for the 3G-services by NTT DoCoMo. FOMA MOAP is a closed platform, meaning that no third-party C++ applications can be installed [15].

3.3.2 Application Development

The official Symbian OS guide (v9.3)[13] names the following choices of programming languages available for the developer:

[12] http://www.uiq.com/pressreleases2007_2824.html, last viewed 2008-01-31

[13] http://www.symbian.com/developer/techlib/v9.3docs/doc_source/guide/cpp-intro/ProgLanguages.html, last viewed 2008-01-31

- Symbian C++
- Java
- Assembler-Code
- C
- OPL
- Web Development
- Other (Python, ...)

From this list of technologies only Symbian C++, Java and mobile web development are considered real options for application development on Symbian.

Just as for desktop applications, Assembler is only needed in very rare situations where performance is of utter need and not even C/C++ performs sufficiently well enough. As a consequence Assembler is only used for few operating system tasks and never needed for application development.

Although with some omissions Symbian OS supports the C Standard Library, it would normally not be considered for new application development but for situations where existing code should be ported to Symbian OS.

OPL (Organiser Programming Language or Open Programming Language) is a Basic-like dialact that originates from the Psion PDAs. Characteristical for a representative of the Basic family of languages, it has a shallower learning curve than C++ but also does not provide its performance and wide range of possibilities. Since version 6 of Symbian OS the language is Open Source under a LGPL-licence. Just like Symbian C++, OPL development ties the application developer to the Symbian platform while at the same time not having the performance benefits and rich set of the possibilities that C++ offers. Similarly to other languages like Python, OPL is not by default available on every Symbian enabled device which creates another level of fragmentation and makes a large deployment of applications difficult.

Symbian C++

As the Symbian OS is itself written in C++, it is therefore considered the primary programming language. As a typical representative of an operating systems native language, C++ offers the greatest possibilities and best performance regarding memory footprint and execution speed. C++ offers full access to every exposed library and is required to be used for the development of servers, plug-ins that extend a framework and device drivers that

interact with the kernel. The logical consequence of using Symbian C++ is of course a tight coupling to the Symbian operating system.

The division of the Symbian devices into three different UI platforms naturally affects native application development with C++. Since MOAP is a closed platform, MOAP Symbian devices can not be targeted by 3rd party C++ application developers, leaving the Symbian UIs divided into S60 and UIQ.

Nokia's S60 is neither binary- nor source-code compatible with SonyEricssons UIQ. This means that an application compiled for S60 can not be run on a UIQ device. Also the source code of an S60 program can not be directly compiled into an UIQ application. Of course this applies to both directions, so the same is true viewed from the UIQ position.

But as both UI platforms are layers on top of the same operating system, the effort to develop an application that runs on both UI platforms can be reduced to an acceptable minimum. Generally a clear separation of directly Symbian related "engine code" and UI proprietary code is advised. This way all the algorithmic logic can be jointly used by the S60 and UIQ application. The code separated in such a way can then be individually compiled for the desired UI platform. There also exist tools that allow for automatic porting of UIQ or S60 source code to the respectively opposite platform.

Symbian Signed In response to security threats to earlier versions of the Symbian OS, a new security mechanism was introduced with release of version 9. The heart of this new security architecture is a certification mechanism. The operating system APIs have been logically regrouped into three different security levels: unsigned, signed and phone manufacturer approval required APIs (see figure 3.5 [13])

Symbian OS v9 is based on a "least privilege" security model, which per se grants applications only minimum access to resources [13]. Functionalities which might pose a security threat when abused need to have certain *capabilities* to be used. These capabilites are granted by certificates. The available certificates range from simply self-signed certificates to certificates signed by Symbian or one of its official test houses, which may involve the approval of the device manufacturer. Alongside with the granted capabilities the costs for certification increase and the level of authentication varies from mere developer authentication to full source code inspections.

Due to the significant architectural change introduced with the new security model, older applications running on Sybmian OS up to version 8 are not binary compatible with version 9 [13]:

The essential architectural changes in Symbian OS v9 mean there

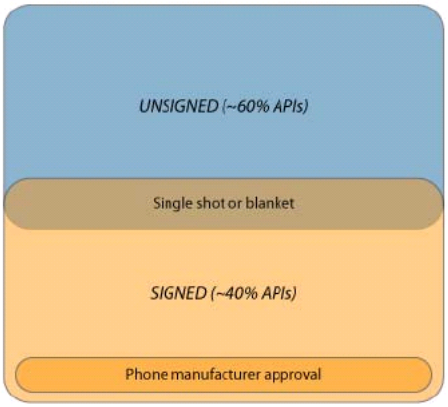


Figure 3.5: Symbian OS security levels

> *is no direct binary compatibility with earlier releases of Symbian OS, however a large degree of source code compatibility has been maintained where possible.*

Java in Symbian OS

Like figure 3.4 on page 26 depicts, Java is an integral part of the Symbian operating system architectue. Java integration is based on standard Java ME with the CLDC 1.1 and MIDP 2.0. MIDP 2.0 is available since Symbian version 7.0 and the CLDC 1.0 from earlier realeses was superseded by the CLDC 1.1 with version 8 of the OS.

According to the official operating system guide[14], the following optional JSRs are currently supported by Symbian OS:

- JSR 82: Java APIs for Bluetooth – v7.0s (Bluetooth Push was added in v8.0).
- JSR 120: Wireless Messaging API – v7.0s.
- JSR 185: Java Technology For The Wireless Industry (JTWI) – v7.0s (this was developed for v8.0, but was backported to v7.0s).
- JSR 139: CLDC 1.1 – v8.0.
- JSR 75: FileConnection Optional Package – v8.0.
- JSR 135: Mobile Media API – v8.0.
- JSR 75: PIM Optional Package – v8.1.

[14] http://www.symbian.com/developer/techlib/v9.3docs/doc_source/guide/J2ME-subsystem-guide/JavaMIDP/OptionalPackages.html, last viewed 2008-01-31

- JSR 184: Mobile 3D Graphics API – v8.1.

The device manufacturers are free to remove any of the optional APIs listed here, as well as to add aditional functionalities.

3.4 Microsoft - Windows Mobile

The Windows Mobile family of operating systems is Microsoft's contribution to the world of small mobile devices. The most current version is Windows Mobile 6, which comes in three flavours:

- **Windows Mobile 6 Classic**
  (Formerly: Windows Mobile for Pocket PC)
- **Windows Mobile 6 Professional**
  (Formerly: Windows Mobile for Pocket PC Phone Edition)
- **Windows Mobile 6 Standard**
  (Formerly: Windows Mobile for Smartphone)

According to Microsoft's Developer Network MSDN[15] the product names have been changed to better reflect the realities of today's mobile device marketplace, where former distinctions between different classes of devices blur rapidly and the smartphone is becoming the universal mobile handset.

As Mike Hall, Technical Product Manger in the Windows Embedded Product Group, clarifies in his blog[16], the basis of all types of Windows Mobile is Windows Embedded CE (Windows CE). Windows CE is a hard-realtime component based operating system that can be customized to fit different fields of application. There are over 700 components available on top of Windows CE, ranging from .Net Compact Framework over web server and web browser to media player. It is by default headless, meaning that it does not have a standard user interface (UI). This allows for a broad field of application, ranging from embedded devices like industrial robots, set top boxes or automobilies, to rich graphic mobile phones and pocket PCs. Windows Embedded CE gets licensed to Microsoft external customers, as well as to the Microsoft internal group that builds the Windows Mobile OS.

The underlying operating system technologies and the APIs are consistent across all Windows Mobile devices. The main difference lies in the different form factors of the target devices, e.g. screen resolution and input facilities (QWERTY keyboard, touchscreen, ...). Generally an application

[15] http://msdn2.microsoft.com/en-us/library/bb158525.aspx, last viewed 2007-01-18

[16] http://blogs.msdn.com/mikehall/archive/2007/01/17/windows-mobile-and-windows-embedded-ce-what-s-the-difference.aspx, last viewed 2007-01-18

developed for a particular flavour of Windows Mobile should work across all Windows Mobile devices.

Similar to the BlackBerry (see next section 3.5) productline, Microsoft offers not just the mobile device platform, but provides an entire corporate software infrastructure for tightly integrating the Windows Mobile device into an existing business infrastructure. This is especially usefull for enterprises that are built on a Microsoft infrastructure and already rely on the various Microsoft server solutions. One of the key servers in such a Microsoft business infrastructure is Exchange Server 2003 and 2007. Like the BlackBerry architecture, it can be configured to directly push contents like emails to the client instead of having to be polled pro-actively. It is also possible to remotely control a mobile device's behaviour, by enabling policies and configuring them respectively. These policies are set on the Exchange Server and are delivered to the client through Exchange ActiveSync, each time the user of the client starts synchronization. Through Exchange Server ActiveSync, it is even possible to remotely erase data stored on the device if it gets stolen or otherwise compromised. Data stored on a mobile device can also be secured by setting a policy that forces the device to locally delete all saved data after an administrator-settable number of incorrect passwort entry attempts [37].

3.4.1 Native Application Development

The term *native* in this context refers to applications that are specifically tailored for Windows Mobile and are built with the corresponding available APIs. In Windows Mobile this can either be done by writing C++ native applications, or by developing managed applications.

Figure 3.6 shows an overview of the APIs exposed through the Windows Mobile 6 SDK. All of these libraries are accessible via native C++ applications. The managed part of the Windows Mobile SDK is made up of the Microsoft WindowsMobile class library (Microsoft Mobile managed assemblies).

C++ Native Applications

Native applications are written in C++ and are directly compiled into the machine language of the target processor, which in case of Windows Mobile is the ARM-architecture. Native applications are characterized by their high performance, low level access (can directly address the devices hardware) and low memory footprint, and are therefore specially well suited for time critical applications. The downside of native applications is that they are generally harder to develop and the developer has to take care of things like memory management, type safety or index boundaries.

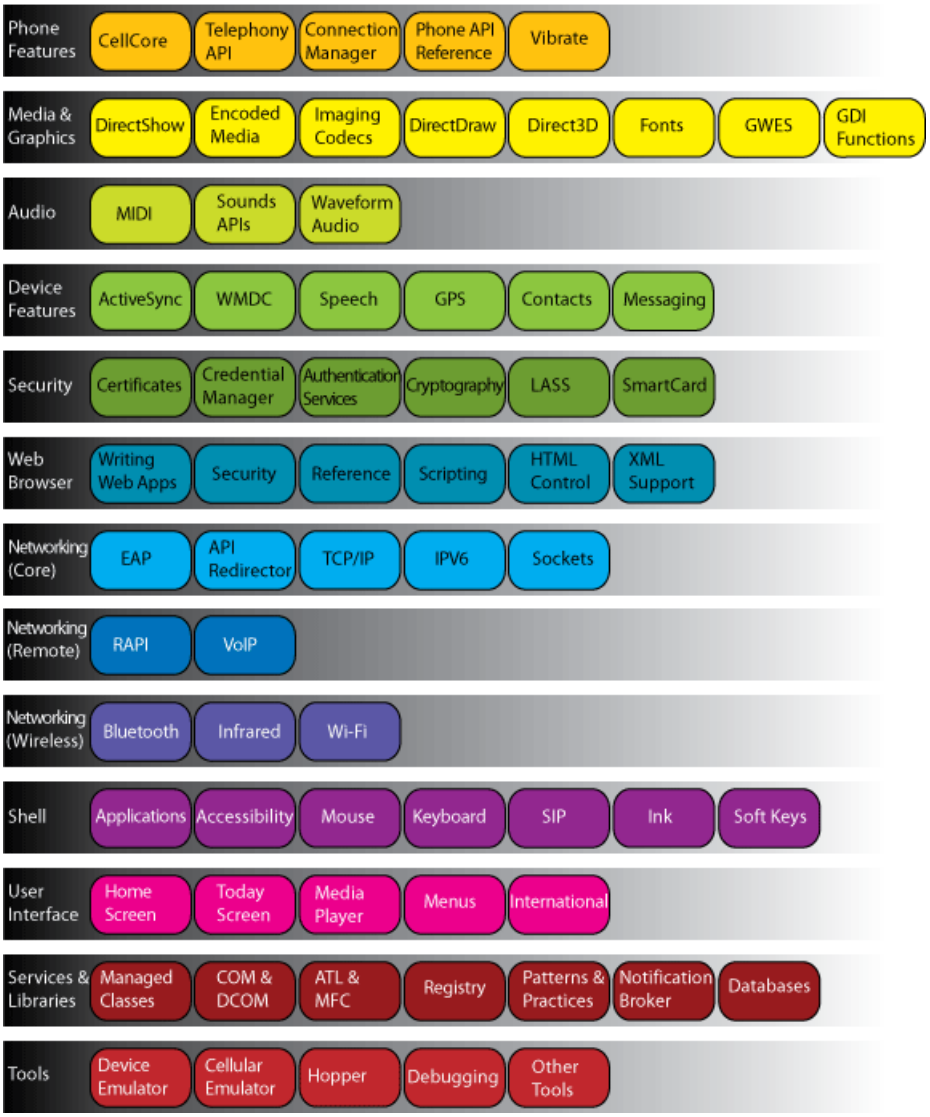


Figure 3.6: Windows Mobile 6 SDK overview

As the Windows Mobile APIs (figure 3.6[17]) are written in native code, one can always forthright use all available libraries when developing native C++ applications. This is opposed to managed applications where the native libraries are provided through a wrapper. In some situations where certain APIs are not provided as a managed library one may be forced to resort to the traditional native Windows Mobile APIs. Native (unmanaged) code can be called from managed code using Plattform Invoke (P/Invoke) [31].

[17] http://msdn2.microsoft.com/en-us/library/bb158486.aspx, last viewed 2007-01-19

Managed Applications

Mobile managed applications are developed using the Microsoft .Net Compact Framework and the Windows Mobile managed assemblies (WindowsMobile Class Library). The most current version of the .Net Compact Framework is version 2. The .Net Compact Framework is a variant of the standard .Net Framework for desktop and server systems, specifically tailored for mobile devices. The architecture of the .Net Framework is similar to that of Java. Applications are not compiled into native machine language but into a platform independent intermediate language (IL) code. When the program is executed, the Common Language Runtime (CLR) performs a Just-In-Time (JIT) compilation of the IL code into the target devices native code. Therefore the CLR acts as a Virtual Machine (VM) for the .Net Framework by managing tasks, like garbage collection, type checking, exception handling and security enforcement [87]. That is why applications written in the .Net Framework are generally referred to as *managed* applications.

Like the KVM running the CLDC in the Java Micro Edition environment, the CLR of the .Net Compact Framework is specially designed for resource constrained mobile devices. The CLR plus the libraries only needs approximately 1.4 MB space in ROM [60].

Microsoft offers a great variety of different programming languages to be used on top of the .Net Framework, with the two most popular (and for the Compact Framework solely available ones) being C# and Visual Basic. Also the .Net Framework comprises a great number of libraries called the .Net class library, ranging from GUI-programming over encryption to web application development. As the .Net Compact Framework is a subset of the standard .Net Framework it implements only approximately 30 percent of the full .Net class library[18]. On the other side it also contains features and classes that are specific to mobile embedded computing and which are not included in the standard .Net Framework.

Some classes are not available in either of the currently available .Net Compact Framework versions 1 or 2 but are exposed through Windows Mobile managed assemblies (WindowsMobile Class Library). They are designed to complement the .Net Compact Framework class library and make up the managed part of the Windows Mobile SDK[19] (see figure 3.6). Windows Mobile managed assemblies can be called from any version (1 or 2) of the .Net Compact Framework and impose a dependency rather on the operating system version than the Compact Framework used. E.g. Windows Mobile 5 introduced many new managed libraries (PIM, telephone interaction, ...) that are accessible via the `Microsoft.WindowsMobile` namespace and can only be used on devices running Windows Mobile 5 (or higher).

[18] http://msdn2.microsoft.com/en-us/library/2weec7k5.aspx, last viewed 2007-01-19
[19] http://msdn2.microsoft.com/en-us/library/bb158492.aspx, last viewed 2007-01-20

The .Net Compact Framework is comparable to a Java ME configuration plus a basic profile (like MIDP). Highly portable applications that do not need to access any device specific functionalities can be written without any additional libraries. The Microsoft WindowsMobile Class Library could be approximately compared to the additional Java APIs. Just like the Java libraries they add aditional functionality mostly targeted to access device specific or lower level features (e.g. PIM, telephone services, GPS, ...). Both libraries have in common that they may not be readily available on the target device.

The ability of calling native functions from within managed code provides the developer with even more possibilities to fully exploit a device's functionalities but increases the dependency for a given version of Windows Mobile.

3.4.2 Java

Windows Mobile does *not* provide a built-in Java Virtual Machine (JVM). Nevertheless some Windows Mobile devices are Java enabled as hardware manufacturers can choose to integrate a JVM to valorise their device. If the device is by default not Java enabled the customer may choose to install a commercial VM that runs on Winodws Mobile like e.g. IBM's WebSphere Everyplace Micro Environment (J9 JVM)[20] or the NSIcom CrEme VM[21].

3.5 Research In Motion - BlackBerry

The Blackberry handheld devices are the most successfull product line of the Canadian based company Research In Motion (RIM). The platform was initially introduced in 1999 as *BlackBerry wireless email solution* [55], comprising BlackBerry Enterprise Server Software, wireless handheld device and radio modem. The BlackBerry family now consists of several different wireless devices with distinct features ranging from PDA to smartphones and different software solutions targeted mostly at business oriented customers.

What sets BlackBerry apart from other mobile platforms is that it is neither a single device nor a single operating system, but a tightly integrated mobile ecosystem comprising handheld device and server software. The application that BlackBerry is most famous for its wireless push services, e.g. push email, push calendar and push contacts and scheduling. Push services allows for directly pushing (sending) data to the handset without having the client to proactively pull it from a server. Instead of the user having to periodically

[20] http://www-306.ibm.com/software/wireless/weme/, last viewed 2007-01-20
[21] http://www.nsicom.com/Default.aspx?tabid=138, last viewed 2007-01-20

check for new messages, with a BlackBerry device, data can be pushed to the customers handset as soon as it is available on the server. Push email is currently regarded as one of the most important use cases for wireless enterprise applications.

Push services to wireless BlackBerry devices is enabled by a special server software that connects the handset to publicly accessible messaging and collaboration systems. This software acts like a proxy that redirects between the mobile device and the corporate software.

Currently there exist two BlackBerry push-service-architectures: The *BlackBerry Enterprise Solution architecture (BES)* (figure 3.7) and the *BlackBerry Internet Service architecture (BIS)* (figure 3.8) [51].

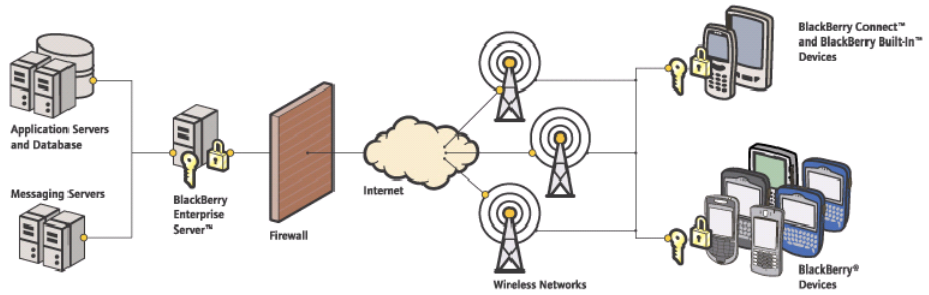


Figure 3.7: BlackBerry Enterprise Solution (BES) architecture

The BlackBerry Enterprise Solution architecture (figure 3.7) is specifically tailored for enterprise environments. Its key component is the BlackBerry Enterprise Server, which needs to be installed behind the corporate firewall. It works as a gateway between the mobile device and the intranet applications of the company that should be exposed to the wireless clients. The server integrates with enterprise messaging and collaboration systems to provide mobile users with access to email, enterprise instant messaging and Personal Information Management (PIM) tools. It can be integrated with IBM Lotus Domino, Microsoft Exchange and Novell GroupWise [22]. By default all traffic between the BlackBerry Enterprise Server and the wireless application on the BlackBerry device is automatically encrypted using Advanced Encryption Standard (AES) or Triple Data Encryption Standard (3DES).

BlackBerry Connect is a software for non BlackBerry devices to connect to the BlackBerry Enterprise Server or BlackBerry Internet Service. It is available for the operating systems Palm OS, Symbian OS and Windows Mobile.

[22] http://na.blackberry.com/eng/ataglance/solutions/architecture.jsp, last viewed 2008-01-11

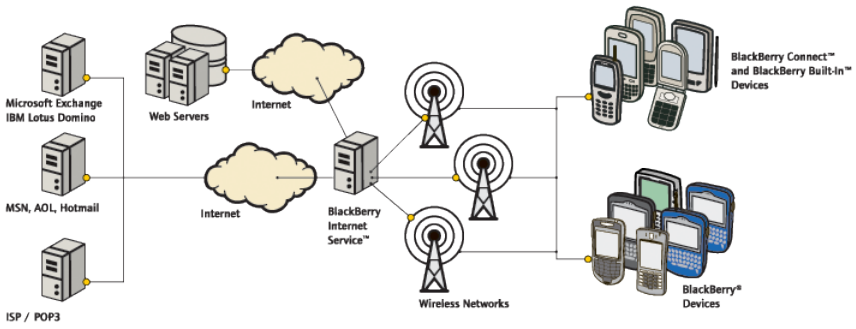


Figure 3.8: BlackBerry Internet Service (BIS) architecture

For private customers and small businesses that can not have an Enterprise Server installed, RIM offers the BlackBerry Internet Service (figure 3.8). The Internet Service architecture offers centrally hosted gateways that allow users to access public email and other internet based applications. Users can either connect to publicly available existing mail boxes or create new ones within the BlackBerry network.

For mobile application development targeted specifically at BlackBerry devices, it is important to know the difference between these two architectures and to be aware of the type of users that should be supported by the application. For the two different platforms there exist different modes of transport and data synchronization.

However due to the large support of the Java Micro Edition on BlackBerry handsets, it is also possible to develop mobile applications that do not rely on the BlackBerry server architectures.

Basically, there are three different ways of developing applications for BlackBerry devices [52]:

- Browser applications
- Java applications
- Rich Media Enhancements with the Plazmic technology

As mobile browsers will be looked at in more detail in chapter 4, they will not be discussed here. Instead the two distinct ways of developing applications for BlackBerry devices via the *Java Development Envirionment (JDE)* and the *Plazmic Content Developer's Kit* will be introduced in more detail.

3.5.1 Java Applications

Although many of the mobile operating systems introduced in this chapter offer Java support in some way, it still has a particular significance for the BlackBerry platform.

Java is tightly integrated within all BlackBerry devices and represents the way with the richest set of possibilities of programming for the BlackBerry platform [51]:

> *It is not possible to write native code for a BlackBerry device, developers are restricted to the Java programming language and the Java APIs that have been publicly exposed on the BlackBerry device.*

As figure 3.9 [50] shows, BlackBerry devices have a proprietary Java Virtual Machine (JVM) which offers both Java ME standard features as well as BlackBerry specific Java API extensions. Java is not just used for third party application development but also for the core BlackBerry applications including email, contacts, calendar, web browser, etc. [51].

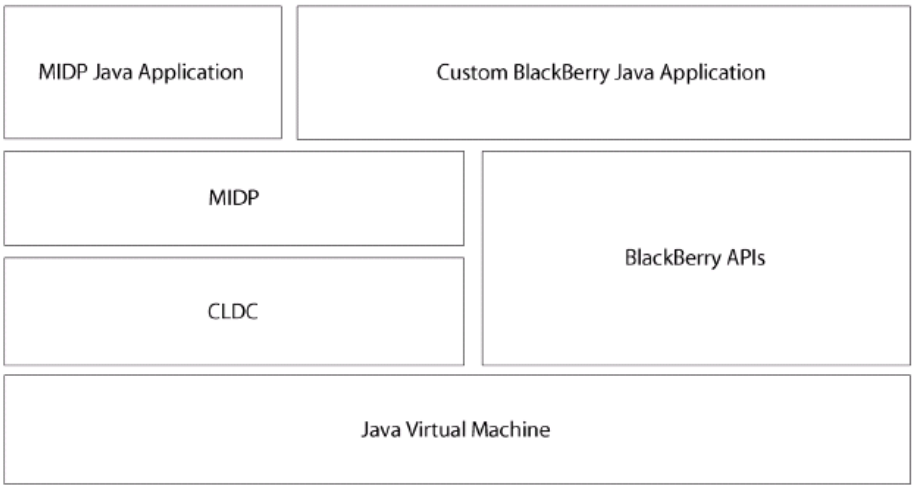


Figure 3.9: BlackBerry handheld software components

Standard Java ME support

All current BlackBerry Java enabled devices support Connected Limited Device Configuration (CLDC) 1.1 (JSR 139) and the Mobile Information Device Profile (MIDP) 2.0 (JSR 118) in the form of Java Technology for the Wireless Industry (JTWI) (JSR 185). As mentioned in section 3.2, JTWI

additionally comprises the Wireless Messaging API (WMA) (JSR 120) as well as the Mobile Media API (MMAPI) (JSR 135).

For address book and calendar access the PIM APIs of the Personal Digital Assistant Profile (PDAP) (JSR 75) are supported. Devices that have an GPS module included, feature the Java ME Location API (JSR 179).

By using only standard Java ME classes, applications can be developed to run on any JTWI enabled device.

In detail the supported APIs are [51]:

- Connected Limited Device Configuration (CLDC) 1.1 (JSR 139)
- Mobile Information Device Profile (MIDP) 2.0 (JSR 118)
- Java Technology for the Wireless Industry (JTWI) (JSR 185)
- Wireless Messaging API (WMA) 1.1 (JSR 120)
- Mobile Media APIs (MMA) 1.1 (JSR 135)
- PIM APIs of the Personal Digital Assistant Profile (PDAP) (JSR 75)
- Location API for Java ME for BlackBerry devices that have a GPS module (JSR 179)

Java API extensions for BlackBerry

For greater support of the particular hardware, BlackBerry also supports a large set of additional Java APIs that are not part of the standard Java ME specification. Although it is not necessary to use these specific classes, they can often provide greater features and functionality over what is supported within the standard Java ME specification, because they are tailor-made for the BlackBerry devices.

The following groups of Java APIs are additionally available for the BlackBerry platform [51]:

- User Interface APIs
- Persistent Data Storage APIs
- Networking and I/O APIs
- Event Listener
- Application integration APIs
  Used to integrate with the existing BlackBerry applications like email, phone, calendar, browser and task list.

- Additional Utilities
  Additional APIs for data encryption and compression, XML parsing, bluetooth connectivity, location based services, etc.

The BlackBerry specific libraries offer extensive support of phone-related features. It is e.g. possible to view, add or change user's contacts in the address book, initiate phone calls, change phone options or to manipulate call logs. Usage of these APIs is only available for signed applications. Every developer can obtain a set of keys for signing its applications at the RIM BlackBerry website[23]. These keys are basically free of charge but an administration fee of USD 100 (at the time of writing of this paper) has to be payed. The actual application code will not be checked by Research In Motion. The certificates are only for identifying the author of an application and thus trying to avoid malicious software by anonymous programmers.

BlackBerry applications can be developed as standard Java ME applications extending the `MIDlet` class defined in the MIDP specification. This is opposed to BlackBerry specific programs which are built as CLDC Applications by extending the class `UiApplication`. Whereas MIDlets can be ported to any device featuring the used Java APIs, usage of BlackBerry UiApplications is restricted to the BlackBerry platform and devices.

Besides of the extended possibilities offered by the BlackBerry specfic classes, the benefits of using the CLDC Application approach over traditional Java MIDlets include:

- MIDlets can not use the BlackBerry User Interface (UI) APIs which are tailor-made for the BlackBerry devices
- MIDlets can not be designed to automatically start in the background when the device powers on
- BlackBerry CLDC Applications can run active background threads even if the application has been closed by the user
- CLDC Applications can exchange information with other applications through inter-process communication

3.5.2 Rich Media Enhancements (Plazmic technology)

The Plazmic technology provides a way of enhancing custom BlackBerry Java or browser based applications with rich multimedia content. With the Plazmic Content Developer's Kit (CDK) content developers can create animated graphics and multimedia screens in the Scalable Vector Graphics

[23] http://www.blackberry.com/developers/downloads/jde/api.shtml, last viewed 2008-01-13

(SVG) standard. By using the CDK the content creators don not require Java programming or web development skills. The media enhancements are created with an application similar to Adobe Flash. Graphical objects can be placed on a timeline and modified (e.g. rotated, scaled, translated, etc.) as the playback advances. In fact, a subset of Adobe ShockWave Flash (SWF) files can even be converted into the SVG format [53].

The SVG-applications can be viewed in the BlackBerry browser as direct downloads from the internet, or embedded in a custom BlackBerry Java application. With the classes provided in the `net.rim.plazmic.mediaengine` packages the Plazmic content can be loaded from local storage or the network and played within the custom application. The possible applications created with the Plazmic technology range from animated films as well as games to entire themes that may customize the look of the BlackBerry device Home screen, dialogs, menus and the idle screen [53].

3.6 Access - Palm/Garnet OS

The Access Garnet Operating System was formerly known as Palm OS and developed by Palm, Inc. With the introduction of the Palm Pilot in 1996, Palm greatly helped evolve the PDA market. Although there have been other PDA devices available by the mid 1990ies, what set the Palm Pilot apart from its competitors, was its ease of synchronization with desktop computers. It allowed the users to synchronize with their desktop applications by just pressing a single button. The Palm devices soon became very popular in the following years and have become the defacto embodiement of the Personal Digital Assistant (PDA) device [87].

The operating system for the Palm devices has ever since been Palm OS. Versions up to 4 have been built for Motorolas DragonBall-processors from the 68,000 (68K) family of processors and did not support multitasking. Starting with version 5, Palm OS now is designed for ARM processors and features multitasking support. For keeping compatibility with applications developed for older Palm OS versions, Palm OS 5 (and higher) offers the built-in Palm Application Compatibility Environment (PACE) which provides a 68K application environment that is equivalent to Palm OS 4.1[24].

After the acquisition of Handspring, Inc. in 2003, Palm was divided into PalmOne and PalmSource. PalmSource continued the development of the Palm OS whereas PalmOne was the manufacturer of the Palm handheld devices [87]. In September 2005 the Tokio based company ACCESS Ltd. announced that it acquired the rights of the Palm OS from PalmSource[25].

[24] http://www.access-company.com/developers/documents/docs/palm_os_garnet_simulator53/Simulator_Concepts.html, last viewed 2008-01-17

[25] http://www.access-company.com/news/press/PalmSource/2005/090905_access.html,

In 2005 PalmOne changed its name back to Palm and now licenses the Palm OS from ACCESS Ltd. Due to the fact that Palm, Inc. holds the rights to the name Palm[26], Palm OS Garnet is now officially called ACCESS Garnet OS, with only Palm Inc. being entitled to rebrand their (slightly modified) version under the name Palm OS.

Presently the most current version of the ACCESS Garnet OS readily deployed on a device is 5.4. Because of the major differences between version 4 and 5 and the already large diffusion of Palm OS 5 (introduced in June 2002), this document will only refer to ACCESS Garnet OS version 5.4.

Palm Inc. is not the only company licensing ACCESS Garnet OS. A complete list of Garnet OS powered devices can be found at the company's website[27]. On the other side not all Palm devices are still based on Palm OS. Since 2005 some Palm handsets are powered by Windows Mobile. At present two of the nine available Palm devices were based on Windows Mobile[28].

3.6.1 Native application development

With more than 20,000 applications[29] available, the Garnet OS is the mobile operating system with the industry's largest collection of third-party software.

Since its appearance in 1996 Palm OS is already available in its 5th generation. Naturally the operating system matured with the capabilities of the devices it was deployed on. To minimize a seemingly inevitable fragmentation of the platform arising from the different versions, Palm OS applications are binary compatible with each other [82]:

> *If you write a brand new application today, it can run on all versions of the operating system provided the application does not use any new features. In other words, if you write your application using only features available in Palm OS 1.0, then your application runs on all handhelds. If you use 2.0 features, your application won't run on the earliest Palm Powered handhelds, but it will run on all others, and so on.*

Even after the switch from the Motorola DragonBall 68K processor to the ARM processor architecture introduced with version 5 of the operating system, this compatibility remains intact. As figure 3.10 [82] shows this is achieved by the Palm Application Compatibility Environment (PACE) that

last viewed 2008-01-17

[26] http://www.palm.com/us/company/pr/2005/052405b.html, last viewed 2008-01-17

[27] http://www.access-company.com/products/accesspowered/index.html, last viewed 2008-01-17

[28] http://www.palm.com/us/products, last viewed 2008-01-17

[29] http://www.access-company.com/products/garnet/whygarnetos/index.html, last viewed 2008-01-17

sits between the core Palm OS libraries and the applications. PACE emulates the 68K-family processor allowing older Palm applications to run without recompilation within the new environment. Due to its well established development paradigm even new Garnet OS applications are commonly still created as 68K applications.

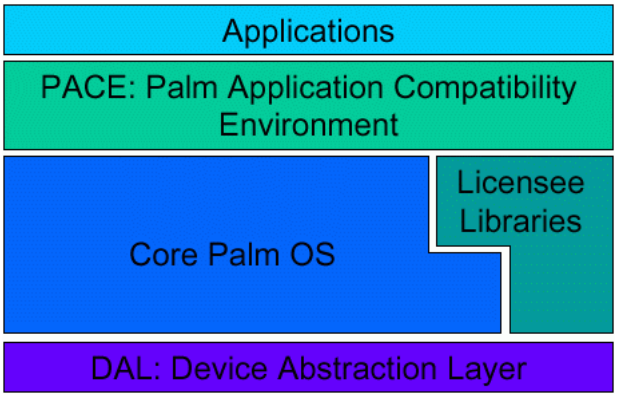


Figure 3.10: Palm Application Compatibility Environment (PACE)

As a consequence of the processor architecture change there are three different types of native Garnet OS applications [45]:

- Palm OS 68K Applications
- PACE Native Objects
- Palm OS Protein Applications

All three types have in common that they are developed with the C/C++ programming language. Although there is support for other programming languages, C is most widely used for Palm OS software development.

Palm OS 68K Applications

Palm OS 68K applications are guaranteed to run on all Palm OS enabled devices. As the name suggests these applications are compiled for the Motorola DragonBall 68K processor architecture. Up to version 4 of Palm OS these applications will natively be run by the device processors. As from version 5 the 68K applications will operate in the Palm OS Compatibility Environment (PACE). As the number of libraries grew along with the version of the operating system, one of course has to take into account the available APIs when trying to target a specific platform. For still being able to use newer APIs while trying to address older versions of the operating systems, there exists the PalmOSGlue library. The PalmOSGlue library runs on Palm OS

2.0 and later. When a PalmOSGlue function is called, it either uses the appropriate function available in the ROM, or if the function doesn't exist, executes a simply equivalent of it [82].

According to the official Garnet OS SDK Documentation, the "Palm OS Programmer's Companion Volume I" [82], most Garnet OS applications are developed as 68K applications, as they do not need extended libraries or code that is natively optimized for the ARM-processor. If the 68K application does not perform adequately well, then there is the chance to optimize the code for the most time critical parts of the application by employing PACE Native Object calls.

Even with calls to the native libraries of the new processor architecture, Palm OS 68K applications follow the development paradigm of the very first Palm application back in 1996, with the most notably drawback being that they can not be multithreaded.

PACE Native Objects

PACE Native Objects (PNO) are somewhat in-between the traditional 68K and the newer Protein applications. They follow the same design principles and are therefore bound to the same limitations as the 68K applications, but can take advanatage of the processing power the new ARM-architecture offers. With the `PceNativeCall()` function a developer can make ARM-native calls to clearly identified segments of the 68K application that need the performance impact the ARM-processor may offer.

Palm OS Protein Applications

Garnet OS Protein applications can take full advantage of all new facilities introduced with the new Palm OS version 5. Some of the most important enhancement include: multithreading, extended database support and a new multimedia framework. Protein applications are compiled for the ARM-architecture.

3.6.2 Java

A Java Virtual Machine (JVM) is not part of the Garnet OS architecture and also not included on Garnet OS powered devices by default. However Palm offers a free download of the WebSphere Everyplace Micro Environment from IBM for its newest Palm OS powered devices[30]. Customers with compatible but older devices may download the WebSphere Everyplace Micro Environment for a small fee[31].

[30] http://www.palm.com/us/support/jvm/, last viewed 2008-01-18

[31] At the time of writing of the thesis the fee was USD 5.99

The WebSphere Everyplace Micro Environment is IBM's version of the Java Micro Edition (ME), centered around their mobile VM J9. The J9 VM basically supports CLDC as well as CDC. The version available as download for the Palm Garnet OS supports CLDC 1.1 with MIDP version 2.

3.7 Apple - iPhone OS

Since its official release on June 29 2007[32], no other single mobile device has recently gained as much attention as Apple's iPhone. With a strong emphasize on its software applications and a new desktop application like mobile web surfing experience, many people see the iPhone at the spearhead of the new wave of mobile computing. Its combination of phone, business applications (organizer, mail, ...) and consumer device (iPod) makes it a prototype represantative of a smartphone as the universal mobile terminal.

Until the official release of the iPhone SDK on March 6 2008[33] little information about the iPhone operating system has been known to the public. Only two distinct devices, the iPhone and the iPod touch [6], use the iPhone OS as their native operating system. As both devices share the same touch centric form factor, the iPhone OS is highly specialized for this very kind of device.

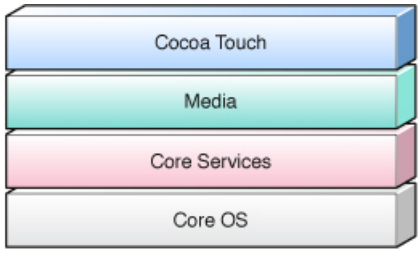


Figure 3.11: iPhone OS technology layers

As figure 3.11 from the iPhone developer website [6] shows, the Cocoa Touch layer is an integral part of the iPhone OS. Cocoa is the application (development) environment for both the Mac OS X and the iPhone OS [5]. It consits of software libraries, a runtime and an integrated development environment (IDE). The Cocoa libraries are mostly written in Objective-C, so native application development for the iPhone also is done with Objective-C.

[32] http://www.apple.com/pr/library/2007/06/28iphone.html, last viewed 2008-04-03
[33] http://www.apple.com/quicktime/qtv/iphoneroadmap/, last viewed 2008-04-03

The kernel used in the iPhone OS is a variant of the March Kernel as it is contained in Mac OS X. The lower layers and the core system architecture are also similar to that in Mac OS X. The Core OS and the Core Services layer contain the fundamental interfaces for iPhone OS, like POSIX threads, UNIX sockets or a SQLite database. The interfaces on these layers are mostly C-based [6].

The Media layer is a mixture of C and Objective-C classes and provides interfaces for 2D and 3D drawing, audio, video and OpenGL ES amongst others [6].

3.7.1 Web Application Development

Web application development is particularly important for the iPhone as it has been the only way of iPhone application development before the release of the SDK[34]. The browser on the iPhone is an Apple Safari, which uses the same Web Kit engine as its desktop counterpart. This makes it a remarkable effective mobile browser with support for all the latest modern web standards, such as [8]:

- HTML 4.01
- XHTML 1.0
- CSS 2.1 and partial CSS3
- ECMAScript 3 (JavaScript)
- DOM Level 2
- AJAX technologies, including XMLHTTPRequest

More information on mobile web applications is provided in chapter 4.

3.7.2 Native Applications

Native application development for the iPhone is possibly since the release of the SDK in March 2008. It is done on basis of the Cocoa application environment together with the Objective-C language. The SDK offers many features for software development that are not available from within a web application [7]:

- Access to low-level features such as threads, ports, and standard I/O
- Support for handling Multi-Touch events
- Support for security features such as encryption, certificate management, and trust policies

[34] http://www.apple.com/pr/library/2007/06/11iphone.html, last viewed 2008-01-30

- Support for rendering 2D and 3D graphics
- Support for game development
- Access to the accelerometer data
- Support for taking pictures with the camera
- Support for BSD sockets and higher-level socket abstractions
- Access to the user's contact information
- Access to the user's photo library
- Access to the user's current location information
- ...

3.7.3 Java

The iPhone does not support any version of Java [8], but vice president of Java marketing at SUN Microsystems Eric Klein announced, that SUN will be releasing a Java ME based JVM as a free iPhone application developed with the iPhone SDK in 2008[35].

3.8 Mobile embedded Linux

Due to its open nature and the resulting adaptability, Linux has made its way on a variety of embedded devices[36] like mobile phones, PDAs, wireless access points, audio/video entertainment devices, robots, automotive devices, etc.

Embedded Linux systems can either be *hard real time* or *soft real time* [87]. Hard real time systems must respond in a deterministic way every time a relevant event occurs. Soft real time systems are also required to respond in a timely fashion, but a definite answer time can not be guaranteed. Hard real time systems are typically found in vital applications like medicine or automotive controls. Mobile Linux on embedded consumer devices like smartphones fall into the category of soft real time systems.

As diverse as the application range is the architecture of embedded Linux systems. The characteristics common to all embedded Linux systems are [87]:

- A pre-emptive monolithic kernel with suport for multitasking and multithreading.

[35] http://www.infoworld.com/article/08/03/07/sun-iphone-java_1.html, last viewed 2008-04-04

[36] For a list of some currently Linux powered devices see: http://www.linuxdevices.com/articles/AT4936596231.html, last viewed 2008-02-02

- A set of libraries and modules usually provided by the open-source community enabling technologies like Bluetooth, wireless LAN, or Graphical User Interfaces.
- Little or no licensing fees and full access to source code through the use of public and open licenses.

The diversity of embedded Linux systems means that application development differs significantly from system to system, as the conceptual similiarities are usually found at a low operating system level. The GUI toolkits greatly affecting the application programming-paradigm and -language range from GTK+ (GIMP-Toolkit) over QTEmbedded to other solutions like the Open Handset Alliance (OHA) Android SGL.

This fragmentation is currently one of the biggest problems of mobile embedded Linux as it discourages third-party developers to enrich the platform with custom applications.

Lately several attempts have been made to consolidate the fragmented Linux platform and to create a consistent application framework that runs applications accross different Linux-based devices. These attempts come from non-profit standard groups like the Linux Phone Standards Forum (LiPS), the Linux Foundation (formerly OSDL) or the Consumer Electronics Linux Forum (CELF) as well as from market-oriented industry alliances like the Open Handset Alliance (OHA) or the Linux Mobile (LiMo) foundation. The non-profit standard groups are involved with the advancement of Linux as an open source platform for mobile devices and the standardization of mobile Linux APIs. They do not provide direct marketable Linux based mobile software stacks. The industry alliances in contrast provide a readily deployable Linux based ecosystem that can be licensed and incorporated by handset manufacturers right away.

3.8.1 MontaVista - Mobilinux

According to the MontaVista website [37] Mobilinux is currently the most widely deployed mobile embedded Linux operating system. It is used in 90 % of Linux-based smartphones with overall more than 35 million phones and other mobile devices running on it. As a commercial-grade Linux platform its sources are not publicly available.

Figure 3.12[38] depicts the architecture of the Mobilinux platform. The underlying graphics toolkit is GTK. Application layer technologies like web browser, Personal Information Manager (PIM) or a Java Virtual Machine (JVM) are not directly provided by the Mobilinux ecosystem. They are

[37] http://www.mvista.com/product_detail_mob.php, last viewed 2008-02-04

[38] http://www.mvista.com/downloads/MontaVista_Mobilinux_5_datasheet.pdf, last viewed 2008-02-04

rather added by the licensees as third party add-ons. MontaVista has recently joined the ACCESS Connect Ecosystem (ACE) partner program [39]. The Japanse based company ACCESS which develops the ACCESS Palm/-Garnet OS also provides a Linux-based mobile phone architecture (see next section).

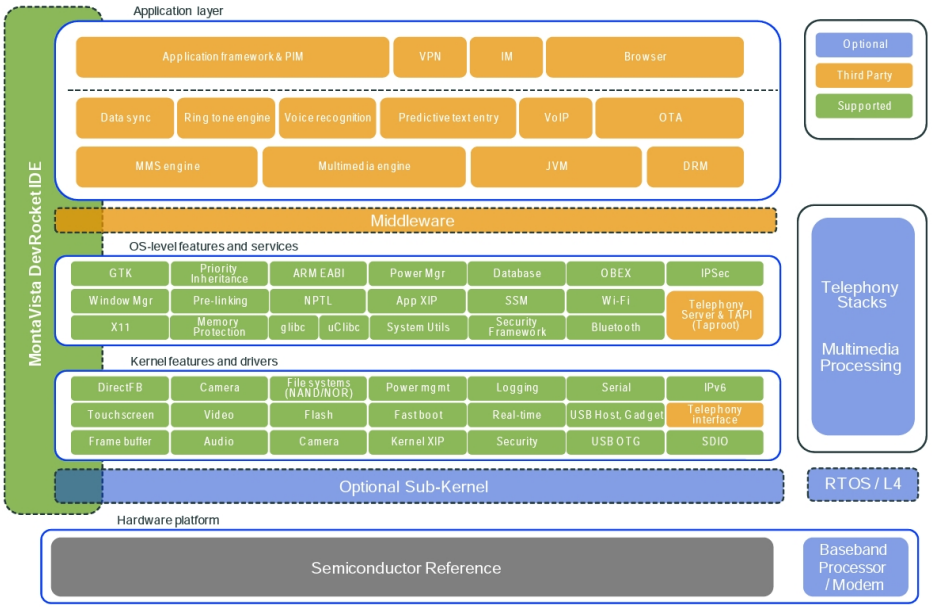


Figure 3.12: MontaVista Mobilinux architecture

3.8.2 ACCESS - ACCESS Linux Platform (ALP)

The Japanese company ACCESS that has acquired PalmSource, provider of Palm OS, in 2005[40] is developing a Linux-based mobile ecosystem called *ACCESS Linux Platform (ALP)* (see figure 3.13[41]).

Presently no ALP based handset has yet been commercially deployed. According to market research institute OVUM[42], ALP is expected to be used in a handset by mobile network operator Orange due in the first half of 2008.

The ALP is designed to replace the existing Garnet/Palm OS. Due to a compatibility layer all legacy Palm OS applications are expected to run

[39] http://www.access-company.com/news/press/ACCESS/2007/20071023_montavista.html, last viewed 2008-02-04

[40] http://www.access-company.com/news/press/PalmSource/2005/111405_access.html, last viewed 2008-02-04

[41] http://alp.access-company.com/files/ALP2007_08.pdf, last viewed 2008-02-04

[42] http://www.ovum.com/news/euronews.asp?id=6472, last viewed 2008-02-04

on the new platform without recompilation. Capable of running Palm/Garnet OS, Java, and GTK+ native Linux based applications, the ALP stack promises to support a huge application base. As the focus of the ALP is primarily on higher level mobile-phone software and the application development stack, ACCESS has recently anounced a partnership with Montavista, of which the focus is predominately on the kernel and the lower level Linux APIs (see previous section).

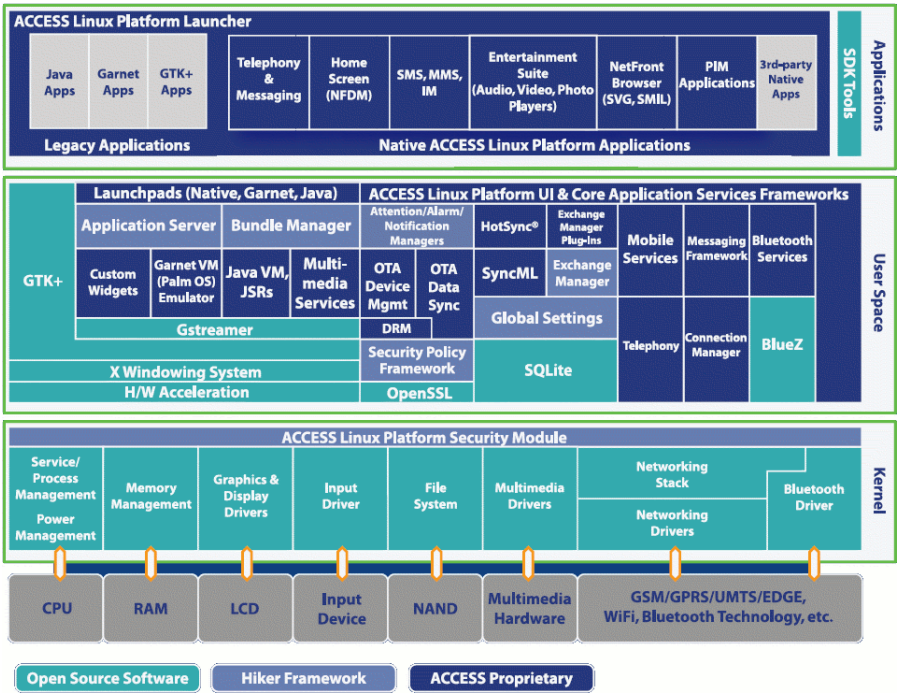


Figure 3.13: ACCESS Linux Platform (ALP) architecture

3.8.3 Open Handset Alliance - Android

Open Handset Alliance (OHA) Android has currently brought huge attention to Linux based mobile systems. Since the OHA is spearheaded by Internet giant Google, the new platform is recently one of the most discussed issues in the realm of mobile computing. It has long been rumoured that Google is about to release its own integrated highend smartphone like Apple's iPhone with unique hardware and a closed operating system. Myths about the alleged "GPhone" have been clarified on November 5, 2007 when Google Inc. officially announced the Open Handset Alliance with its An-

droid mobile platform[43]. Instead of a single integrated mobile device, OHAs Android is a Linux-based operating system and software stack which will be deployed on many different devices from various handset manufacturers. Its goal is to consolidate the fragmented mobile Linux community and to provide a pseudo-standard application development environment which runs Android applications across a wide range of different handsets. Aside from Google the OHA counts over 30 founding members ranging from semiconductor companies, handset manufacturers, mobile operators to software and commercialization companies. A complete list of the current members can be found at the OHA website[44].

Currently no Android powered device has yet been released. According to a Google press release[45], the first handsets built on the new platform will become available in the second half of 2008.

Architecture

Figure 3.14[46] depicts the major components of the Android operating system.

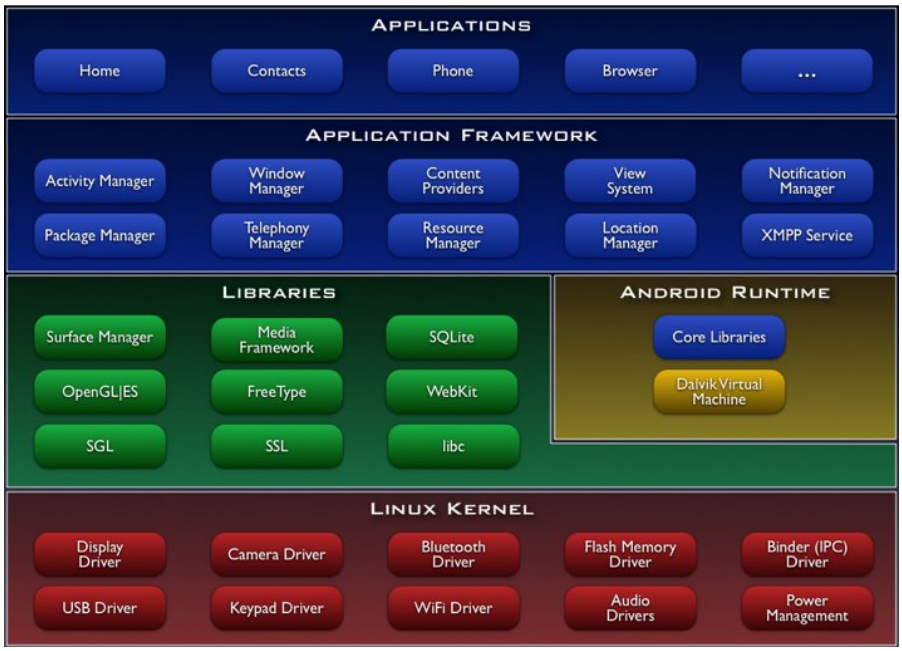


Figure 3.14: Android architecture

[43] http://www.google.com/intl/en/press/pressrel/20071105_mobile_open.html, last viewed 2008-02-07

[44] http://www.openhandsetalliance.com/oha_members.html, last viewed 2008-02-07

[45] http://www.google.com/intl/en/press/pressrel/20071112_android_challenge.html, last viewed 2008-02-07

[46] http://code.google.com/android/what-is-android.html, last viewed 2008-02-07

As with any other mobile embedded Linux platform, the *Linux Kernel* forms the basis of the Android architecture. Android builds on a 2.6 Linux Kernel for core system services like security, process and memory management, network stack or driver model.

The layer above represents the *native libraries*. The green color indicates that these libraries are written in either C or C++. They provide typical core library functionalities like classes for basic GUI operations, media codecs, security certificates or font rendering. Just like the Safari browser used on the iPhone, the browser embedded in the Android OS is also built on the Open Source Webkit browser-engine.

The next component in the Android operating system the *Android Runtime* depicts something very untypical for Linux based mobile systems. It is a special Java Virtual Machine responsible for running all the applications in the Android environment. No applications, neither the pre-installed core applications like email, SMS or calendar, nor any third-party applications can be written in native C or C++, but have to be developed using Java[47]. This way the VM serves as an abstraction layer between the core system components and the application development framework. This bears some significant similarities to the BlackBerry platform. What completely sets Android apart from other Java based platforms is its VM called *Dalvik Virtual Machine*, which is not at all compliant with the reference SUN JVM, or KVM.

The main differences are[48]:

- The Dalvik VM is register-based and not stack-based.
- The Dalvik VM uses a proprietary byte-code format called Dalvik Executable (.dex).
  In order for applications compiled on the desktop with a standard Java VM to be run on Android, their bytecode first has to be transformed into the .dex-format. This is done with the help of the *dx* tool included in the Android SDK and can be made by hand or automated at build-time.
- The libraries are not standards conform.
  The core libraries used by the Android Runtime are neither Java ME nor Java SE compliant. The bigger part of the standard Java APIs are Java SE libraries. From the Micro Edition, only the Java binding for OpenGL ES is implemented. Other libraries contained in the Android platform come from Open Source projects like the Apache Commons or JUnit as well as Google. All in all the by default included packages should be more familiar to desktop than to mobile application

[47] http://code.google.com/android/kb/general.html#c, last viewed 2008-05-21
[48] http://code.google.com/android/what-is-android.html, last viewed 2008-02-07

developers. A complete list of packages can be found on the Android website[49].

As stated in the official online documentation[50], every Android application runs its own process with its own instance of the Dalvik Virtual Machine.

The last building block of the Android architecture is the *Application Framework*, which is the direct basis for built-in as well as third party applications.

Standard Java and Java ME applications

Despite of the fact that all Android applications are written in Java, the platform is neither compliant with Java SE nor with any of the two Java ME profiles (CLDC and CDC). To run existing Java applications on top of Android all the same, the currently most promising approach is probably to port them.

Open Handset Alliance founder member Esmertec has announced that its commercial embedded Java Virtual Machine *Jbed* can be made available to members of the OHA on demand[51]. This means that handset manufacturers can choose to integrate the Jbed VM into their Android platform stack and in such a way make some Android phones Java ME enabled out of the box.

3.8.4 Others

Besides the already mentioned initiatives, many other manufacturers, software development companies and consortiums are currently developing or already have developed mobile Linux platforms. Amongst others they comprise:

- **LiMo Foundation**
  (http://www.limofoundation.org/)
  The LiMo Foundation was founded at the beginning of 2007 by hardware manufacturers and network operators Motorola, NEC, NTT DoCoMo, Panasonic, Samsung and Vodafone [29]. Its mission is to create an open, Linux based mobile middleware platform which can be extended and customized by mobile stakeholders like device manufacturers and network operators in a uniform way. As a middleware provider, the LiMo platform offers the Linux Kernel and all essential operating system and user interface components. The lowest layer including device drivers and hardware interfaces, as well as the topmost application

[49] http://code.google.com/android/reference/packages.html, last viewed 2008-02-07
[50] http://code.google.com/android/what-is-android.html, last viewed 2008-05-22
[51] http://www.esmertec.com/solutions/mobile_multimedia/android_platform/index.shtml, last viewed 2008-02-07

layer, are left open for the foundation members for device and UI design customization and differentiation. The application user interface framework is based on GTK+ [28]. By the agreement of the industry on one standard, LiMo expects lower cost of development and faster time to market for new devices. Software development SDKs for native C/C++, Java and Webkit programming are scheduled to be released in the second half of 2008. By February 2008, 18 handsets from 7 vendors have already been commercially deployed on the LiMo platform [29].

- **Nokia - maemo**
  (http://maemo.org)
  Maemo is an open source mobile platform for Nokia Internet tablet handhelds like the N770, N800 or N810 and other Linux-based devices. Since it does not provide a telephony stack it is not appropriate as a smartphone operating system. The maemo Kernel is a Linux 2.6 Kernel, based on the ARM processor family branch, which can be modified, recompiled and flashed by the developer [41]. It largely relies on the same open source Linux components found in the Debian distribution. The user interface architecture builds on GTK+/GNOME. Its user interface *Hildon UI*, is customized for the screen size and usage scenarios of typical touch screen enabled mobile devices[52].

- **OpenMoko**
  (http://www.openmoko.com)
  OpenMoko is a project of the Taiwanese manufacturer FIC aimed at building an entirely open Linux mobile phone. The reference device is the FIC Neo1973, which comes standard with a GTK+/GNOME based Linux-stack provided by OpenMoko[53]. OpenMoko also posted the CAD (Computer Aided Design) files of the Neo1973 under a creative common license, which provides a completely in-depth look at the internals of the device. At the time of writing of this thesis, the phone was solely aimed at developers with strong system and programming skills[54].

- **SUN Microsystems - JavaFX Mobile**
  (http://java.sun.com/javafx/mobile/)
  JavaFX Mobile is a forthcoming mobile software system based on Linux and Java by SUN Microsystems. It is the successor of the SavaJe rich Java mobile platform from SavaJe technologies, which Sun acquired on May 8, 2007[55]. Its basic architecture is very similar to Open Handset

[52] http://www.forum.nokia.com/main/platforms/maemo/index.html, last viewed 2008-04-05
[53] http://linuxdevices.com/news/NS5429713730.html, last viewed 2008-04-05
[54] http://wiki.openmoko.org/wiki/Developer_preview, last viewed 2008-04-05
[55] http://www.sun.com/software/savaje/index.xml, last viewed 2008-04-04

Alliance's Android. Java serves as the primary application development language and offers full access to the device capabilities. Underneath lies a Linux Kernel and a thin layer of native low-level services and libraries. Unlike Android, JavaFX Mobile will offer full support for Java ME, which means that Java ME applications should work out-of-the-box with little or no porting. At the time of writing of this thesis no JavaFX Mobile based handset, SDK or simulator has yet been officially available.

- **Trolltech - Qtopia Phone Edition**
  (http://trolltech.com/products/qtopia/phone_edition)
  The Norwegian company Trolltech is mostly famous for developing Qt, a multi-platform C++ GUI framework which is used amongst others by the Linux Desktop environment KDE. Qtopia is an application platform based on Qt. Qtopia Phone Edition is an application platform and user interface for Linux powered mobile phones. Besides Qtopia and other native applications it offers a Java integration for running Java ME applications with a native Qtopia look and feel.

  It can be deployed on mobile phones based on Intel's x86 architecture as well as ARM processors. The current minimum Kernel requirement is 2.4. The main audience of Qtopia Phone Edition are device manufactureres that build customized Linux phones on top of it. According to Trolltechs website[56], already millions of Linux phones powered by their application platform have shipped.

  Trolltech also offered a reference mobile ecosystem in the form of the Qtopia Greenphone which was an open full functional Linux mobile device loaded with Qtopia Phone Edition. Trolltech announced that the Greenphone has been sold out and will not be produced any more. Instead Trolltech suggests OpenMoko's Neo1973 as a phone hardware for Qtopia Phone Edition development[57].

  On January 28, 2008, Trolltech was acquired by Nokia[58].

[56] http://trolltech.com/products/qtopia/phone_edition, last viewed 2008-04-05

[57] http://trolltech.com/products/qtopia/greenphone/greenphone_pricing, last viewed 2008-04-05

[58] http://trolltech.com/company/newsroom/announcements/press.2008-01-28.4605718236, last viewed 2008-04-05

Chapter 4

Mobile Web Applications

4.1 Overview

Web applications have successfully proven to be technical enablers for Service Oriented Applications in the realm of desktop computing. Their major advantage over native applications lies in the independence from a particular operating system. The nature of web applications is per se distributed on basis of a client-server architecture. The only requirement for the client is to have a browser installed, which is capable of viewing the contents delivered by the server. Usually web applications adhere to a *thin-client* paradigm, meaning that most of the application logic is done on the server. With advanced programming languages and frameworks available on the server, a contemporary web application usually fulfills various steps by processing a request, involving complex calculations, database access, communication with other servers, etc., before sending simplified content (usually HTML) back to the client. Today's modern web application frameworks overcome traditional drawbacks of web application development, like the stateless nature of HTTP, the tedious composition of HTML, etc.

Also desktop web browsers have matured, providing a strong support for client technologies like (X)HTML, XML, CSS, JavaScript, Java Applets, Adobe Flash, Apple Quicktime, etc. The increasing level of conformance of the browser-manufacturers to web technologies standardized through the World Wide Web Consortium (W3C)[1] allows for writing cross-platform browser applications with little or even no recoding.

With the advent of dynamic lightweight client technologies like Asynchronous Javascript And XML (AJAX), the SOA character of web applications has even been further augmented. Given AJAX's asynchronous nature, it is possible to bypass the static request-response roundtrip of traditional web applications, which allowed client content retrieval only on a "per-page" basis, in favour of a dynamic approach that only fetches the information

[1] http://www.w3.org/, last viewed 2008-02-14

needed. This reduces the bandwidth needed and greatly improves the users' expierence. This class of new web applications is often referred to as *Web 2.0*, a term which was originally coined by publisher Tim O'Reilly[2]. In a nutshell it describes "symmetric" browser based applications, which allow the user to interact and push data to the server to the same extent as receiving information.

Considering all the benefits of modern web applications they seem to be an ideal solution for a mobile Service Oriented Architecture. The fact that third-party applications for the iPhone could have initially only been developed as web 2.0 applications[3] can also be seen as an indicator of the importance of this technology. The greatest promise of mobile web applications over native applications is the potential of not being tied to a specific platform. This is especially valuable in such an unconsolidated market as the mobile handset market. Another major advantage of mobile web applications is the easy deployment. As no software (besides the browser) has to be installed locally on the client, software modifications and add-ons only have to be applied to the server side.

Also, web browsers execute in a "sandbox", which means that they have little to no potential of harming the device.

Though this is a big advantage concerning security, it is also one of the biggest downsides of mobile browser applications by contrast to native applications. They can not directly interact with the devices native APIs and thus can not fully exploit the augmented mobile context awareness. Also web applications rely more on an intact network connection than their native counterparts, as their entire appearance is downloaded from the server.

This chapter provides an overview of the evolution of mobile web technologies and highlights the possibilities of today's modern AJAX based browser applications.

4.2 Traditional Mobile Web Applications

Mobile web applications followed a different path of evolution than desktop applications. This difference was motivated by the resource-scarceness and the network heterogeneity of (former) mobile devices. In the late 1990ies, the wireless industry was seeking for a standard way of delivering Internet data to wireless clients over the existing cellular networks (for a brief overview of cellular networks see 2.2.1).

As a consequence, in June of 1997, Nokia, Ericsson, Motorola and Un-

[2] http://www.oreillynet.com/pub/a/oreilly/tim/news/2005/09/30/what-is-web-20.html, last viewed 2008-02-14

[3] http://www.apple.com/pr/library/2007/06/11iphone.html, last viewed 2008-02-14

wired Planet formed the Wireless Application Protocol (WAP) Forum [1, Chapter 24]. The WAP Forum[4] served as a standards body for creating the Wireless Application Protocol (WAP).

WAP has been the primary enabler of mobile Internet in Europe and the U.S.A. and consists in essence of two building blocks [80]:

- WAP Protocol Stack
- Wireless Application Environment (WAE)

Its primary design goals were to enable delivery of Internet content to mobile devices by abstracting from the underlying bearer technologies (e.g. GSM, GPRS, EDGE, CDMA, ...) and to allow for application development that is similar to traditional web application development, but offers optimizations tailored specifically at the limited target devices. Since the first release of the WAP 1.0 specification in 1998 [80], WAP has meanwhile evolved into WAP 2.4. There are considerable differences between the two major versions of WAP, which reflect the rapid evolution of the wireless mobile landscape of the past few years.

4.2.1 WAP 1.x

Protocol Stack

As shown in figure 4.1 [77], the physical, datalink and network layers of the ISO-OSI reference model are not directly adressed by the WAP protocol stack. These basic network functionalities are provided by the bearer technologies. The WAP protocol stack abstracts from these bearer technologies and can be implemented on top of any wireless network.

The most important protocol in the WAP stack is the Wireless Datagram Protocol (WDP), which acts as the interface between the rest of the WAP protocols and the underlying network. WDP is the equivalent to the User Datagram Protocol (UDP) used in land-based Internet technology. Equally to UDP, WDP offers an unreliable connectionless network service, which was chosen because of the varying connection qualities of wireless networks.

Wireless Transport Layer Security (WTLS) provides a means of secure WAP connections, similar to TLS 1.0. Wireless Transaction Protocol (WTP) provides request-repsonse services to the Wireless Application Environment (WAE), with every request-response pair being one transaction. The final layer before the WAE, the Wireless Session Protocol (WSP) provides HTTP/1.1 functionality to the WAP application layer on top of WTP (connection-mode) or WDP (connectionless).

[4] In June 2002 the WAP Forum was transformed into the Open Mobile Alliance, which today counts nearly 200 company-members. http://www.openmobilealliance.org/AboutOMA/FAQ.aspx, last viewed 2008-02-14

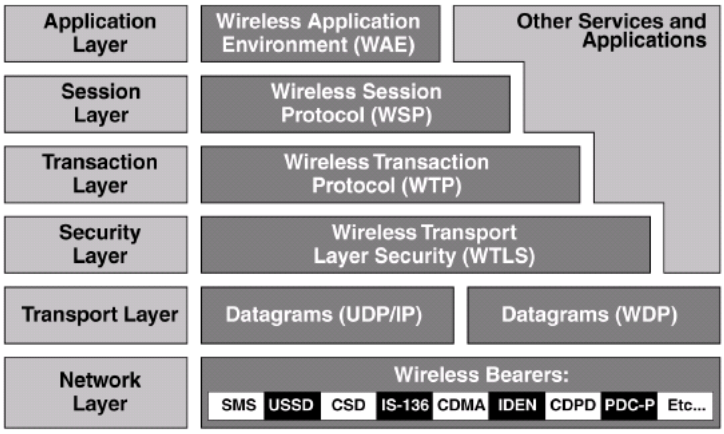


Figure 4.1: WAP 1.0 protocol stack

Application Environment (WAE)

The Wireless Application Environment (WAE) is what is usually associated with WAP, since it is the most important layer for the application developer. It provides a number of different components that enable the development and execution of mobile WAP based Internet applications:

- Wireless Markup Language (WML)
- WMLScript
- Wireless Telephony Application Interface (WTAI) Specification
- Microbrowser Specification

Wireless Markup Language (WML) WML is the markup language used for WAP 1.x applications. It is a true subset of XML, but like all WAP based technologies it is not defined by the W3C, but the WAP Forum. Although it offers familiarities with the standard World Wide Web (WWW) markup language HyperText Markup Language (HTML), they are not compatible. WML clearly reflects its target of mobile devices with very limited resources and small (grayscale) screens. One WML-file consists of a *deck*, that can be devided into several *cards*. A card represents exactly one output screen. Several cards in one deck (=one file) can improve the responsiveness of the application since not for every new screen a page has to be downloaded. Decks can only contain a limited number of cards and are used to group together logical units, like the booking of a ticket. When the last card of a deck is reached, a new deck can be polled from the server. The information from different cards can be collected with the built in *variables*.

Every form element automatically is a variable, but they can also be defined explicitly.

Support for HTML layout-tags like tables, headings, etc. is rather limited or not provided at all. Like in HTML prior to the introduction of Cascading Style Sheets (CSS), layout and design are mixed within the WML markup and not separated. The only image format supported is *wmbp*, which is very lightweight at the expense of image quality. It can only be used for icons and small images like company logos [24].

WMLScript WMLScript is derived from ECMAScript [1, Chapter 24] (=JavaScript) and is used in particular for content-checking of user entered forms as well as datatype conversions and small calculations.

Wireless Telephony Application Interface (WTAI) Through special links prefixed with `wtai://` WAP provides a minimalistic interface to the telephone over WML.

This way, hyperlinks can be used to directly call phone numbers, e.g.:

```
1 <a href=wtai://wp/mc;012345>Call me</a>
```

add entries to the phones addressbook, e.g.:

```
1 <go href=wtai://wp/ap;0123456789,Me/>Add me</go>
```

or send tone sequences, e.g.:

```
1 <go href=wtai://wp/sd;0123456789*\#ABCD/>
```

Microbrowser Specification A WAP 1.x WML-browser is supposed to support the above introduced components of the WAE. Different browsers from different manufacturers may deviate from the specification by the WAP Forum, which is expressed as the *microbrowser specification*. As a consequence, not all the specified functionalities have to be available on all devices. The many minor releases of the WAP 1.x specification with different supported functionalities furthermore created a fragmentation of the application environment.

Programming Model

Just like the Protocol Stack and the Application Environment, also the WAP Programming Model is closely aligned with the standard Web Programming paradigms. As figure 4.2 [80] shows, WAP basically uses a pull-model with request-response roundtrips from the client to the server. Just like any arbitrary web-browser, the WAP client first hast to make a request to the server (pulling), before receiving the requested content. The big difference is that there is a *proxy* or *gateway* [1, Chapter 24] between the server and the client. This WAP gateway is responsible for en-/decoding the WML-contents from

and to the client. The mobile client only sends and receives binary WML and WMLScript, which is smaller than uncompiled sources and thus reduces the communication payload. The gateway is responsible for decoding the data before sending it to the webserver and for encoding the data received from the webserver in order to be viewable by the client.

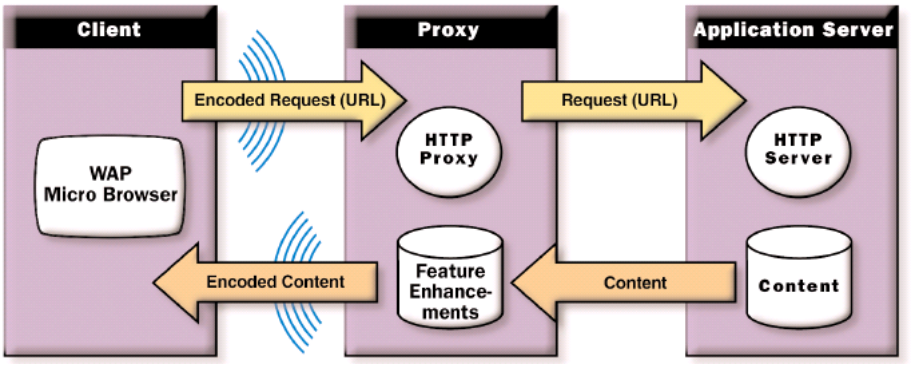


Figure 4.2: WAP 1.0 programming model

Another major responsibility of the WAP gateway is to handle the protocol interworking between the client and the web (application) server. The mobile client talks to the WAP gateway over the protocols defined in the WAP Protocol Stack, as shown in figure 4.3 [80]. The gateway is then in charge of translating the WAP Protocols to their respective land-line Internet pendants, to communicate with the webserver. Responses from the webserver are conversely transformed into the WAP Protocols before being submitted to the mobile client.

The WAP gateway is provided by the mobile network operator[5], so for WAP application development and deployment, a normal webserver housing the WML and WMLScript files is enough. From the perspective of the mobile client, the whole process of communication over the proxy is transparent. To request a specific resource a normal URI over HTTP (e.g. `http://server/resource.wml`) is issued, with the only difference being the `.wml` file-extension instead of the usual `.html`.

4.2.2 WAP 2.x

WAP 2.x, introduced in 2002 [80], greatly reflects the enhancements of mobile communications since the release of WAP 1.0 in 1998. Major design goals have been the convergence with standard web technologies to allow

[5] There are also commercial and an Open Source WAP gateways available, which can be integrated into an enterprise environment. A example for an Open Source gatway is Kanel (http://www.kannel.org/, last viewed 2008-02-15

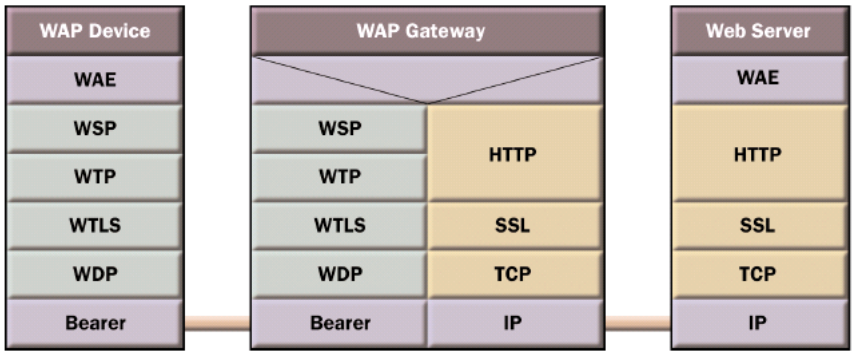


Figure 4.3: WAP 1.0 gateway

the development for one web [46], as well as to remain backward compatibility with the existing 1.x WAP specifications. Also, several new features have been introduced, like the possibility of sending push-content to the client.

Protocol Stack

The WAP Protocol Stack has been considerably changed in WAP 2.x. Like figure 4.4 [80] shows, the legacy WAP 1.x protocols have all been abandoned in favour for new ones that are fully interoperable with existing Internet protcols [80]:

> *A key feature of WAP 2.0 is the introduction of Internet protocols into the WAP environment. This support has been motivated by the emergence of high-speed wireless networks (e.g. 2.5G and 3G) that provide IP support directly to the wireless devices.*

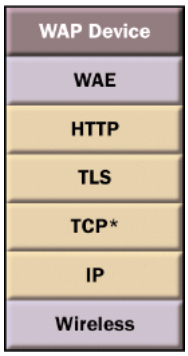


Figure 4.4: WAP 2.0 protocol stack

Application Environment (WAE)

As mentioned above, one of the key goals of the WAP 2.0 specification was to converge mobile and traditional web application development. As a consequence the Wireless Markup Language (WML) used in version 1.x of WAP was abandoned in favour of an XHTML variant that could also be displayed in a normal desktop browser. Additionally separation of layout and style closely aligned to the W3C Cascading Style Sheets (CSS) was introduced. WMLScript completely vanished, which means that by specification there is no client side scripting possibility in WAP 2.x [46][6].

XHTML Mobile Profile (MP) The XHTML MP document type is defined as a strict superset of XHTML Basic [79]. XHTML Basic is specified by the W3C[7] and only includes a minimal set of tags (modules). XHTML MP extends XHTML Basic with modules, elements and attributes to provide an authoring language targeted for resource-constrained mobile web clients.

Because of the close connection between XHTML MP and XHTML (Basic), XHTML MP pages can be displayed in normal desktop web browsers, which greatly eases development of mobile web pages. A complete list of supported tags is provided in the XHTML MP specification [79].

XHTML MP browsers support the common web image formats GIF, JPG and PNG. Together with XHTML MP, style sheets, which allow for the separation of layout and design, were introduced to WAE user agents.

WAP CSS If a WAE user agent supports styling of documents with style sheets, it must support the syling language WAP CSS [79]. WAP CSS is a subset of CSS2 specified by the W3C[8]. As a subset of CSS2, WAP CSS adheres very closely to the W3C CSS in the core functionalities like basic syntax, inheritance, cascading, selectors, box model, etc.

It also supports the following WAP specific extensions [78]:

- **Marquee**
  Properties to create simple animation effects for scrolling text.
- **Input**
  Properties to specify an *input-mask* for the format of the allowed user input of XHTML MP form elements.
- **Accesskey**
  A property similar ot the HTML `accesskey`-attribute that allows for specifying characters for alternative quick navigation.

[6] http://www.developershome.com/wap/xhtmlmp/xhtml_mp_tutorial.asp?page=wmlFeaturesLost#4.5.XHTML0MP\%20Does\%20Not\%20Support\%20Client-side\%20Scripting|outline, last viewed 2008-02-15

[7] http://www.w3.org/TR/xhtml-basic/, last viewed 2008-02-15

[8] http://www.w3.org/TR/REC-CSS2/, last viewed 2008-02-16

WAP CSS specific attributes and attribute-values are always prefixed with `-wap-` (e.g. `a {-wap-accesskey: send, *, #;}` for defining the *send*, * and # button as quick navigation for hyperlinks).

Programming Model

Alongside the employment of standard Internet protocols in the protocol stack, the WAP 2x.programming model has also changed. As figure 4.5 [80] shows, the mobile WAP client device can now directly communicate with the origin webserver. Also the step of en-/decoding the application data drops out, since the XHTML MP and WAP CSS sources are not binary encoded but sent in plain text, like conventional HTML used for desktop web applications.

Still a WAP gateway can be used to offer mobile service enhancements, or for optimizing the communications process. In addition, a WAP proxy is mandatory to offer the newly introduced push functionality [80].

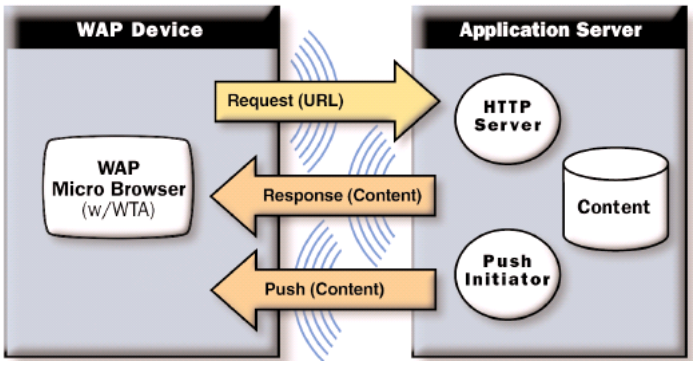


Figure 4.5: WAP 2.0 programming model

4.3 Mobile Asynchronous JavaScript And XML (AJAX)

Modern AJAX based web applications mark the eventual technological convergence of mobile and desktop web development. Together with the evolution of handsets, mobile web browsers mature, offering nearly the same set of technologies to the web developer as their desktop counterparts. This eliminates the need for customized technologies like WAP and allows for the development of *one web*.

4.3.1 The AJAX technologies

There is no technical difference between *mobile* AJAX and AJAX on desktop computers, so in order to understand mobile AJAX it is important to know

what AJAX generally is about.

Rather than being one technology in its own right, like e.g. Java ME or the .Net Compact Framework, AJAX is more a collection of well established client side web technologies, corporately employed in a new way. The term AJAX is shorthand for *Asynchronous Javascript And XML* and was coined by Jesse James Garnet in his online article "Ajax: A New Approach to Web Applications" [34] in February 2005 [35, 86].

AJAX comprises the following basic technologies [34, 35, 86]:

- Standards based presentation: *(X)HTML & CSS*
- Programming and visual object manipulation: *JavaScript & the Document Object Model (DOM)*
- Data Exchange formats: *Plain text, XML, JSON*
- Asynchronous data retrieval over HTTP: *XMLHttpRequest (XHR) object*

All these technologies have been around for years in web development and are commonly found in any modern browser. One of the most interesting technology in the context of AJAX is the *XMLHttpRequest (XHR)* object. It was first introduced by Microsoft in Internet Explorer 5 for Windows as an ActiveX object [3, 86]. XHR liberates web developers from the traditional static page-oriented web application model and enables them to send synchronous and asynchronous HTTP requests with full control over the request and response. While with the traditional web application model the server could only deliver full HTML pages as a response to the web client, AJAX represents a more modular approach. An AJAX client just fetches the information really needed, which avoids the reloading of the client page with every request. Also, the exchanged data format does not need to be HTML, but could rather be any appropriate format.

Figure 4.6 opposes the traditional web application model to the AJAX web application model [86].

It is important to note that for the server there is no difference between an AJAX request and a traditional roundtrip. From a server's perspective it is always normal HTTP communication. The difference is the data format of the response carried over HTTP that could be plain text, XML, or JSON[9] in addition to (X)HTML. The browser takes care of all the steps necessary to provide XHR functions to the content developer – from the initiation of the asynchronous request to the provision of the received data and the call of the callback function defined by the client developer (see figure 4.7 [33]).

[9] JSON (JavaScript Object Notation) is a lightweight data-interchange format for exchanging JavaScript objects as text strings. It can be seen as JavaScript object serialization. http://json.org/, last viewed 2008-02-14

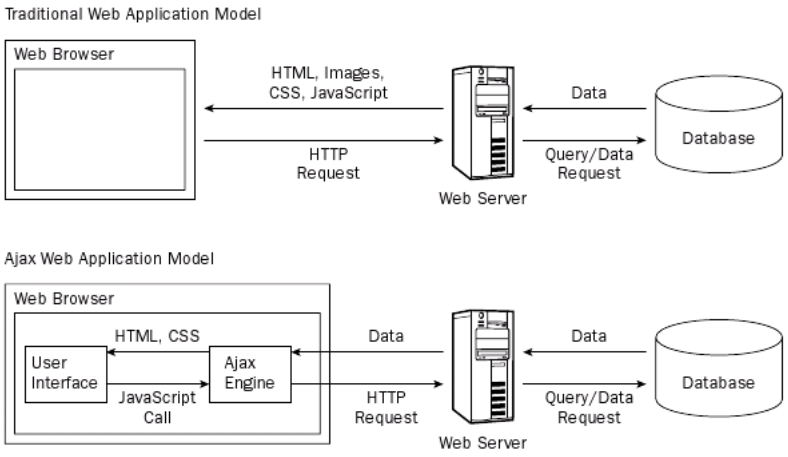


Figure 4.6: Traditional versus AJAX web application model

Due to the asynchrony of XHR calls, control is returned to the client as soon as the request is dispatched. Only when the server has finished the processing of the request, the browser loads the received data into the XHR object and informs the client application by calling the specified JavaScript callback function. This roundtrip behaviour makes AJAX applications extremly responsive compared to traditional web applications.

Overview of the XMLHttpRequest object

The most important members of the XMLHttpRequest object are [36,85][10]:

- *open(http-method, url, async, user, password)*
  Assembles a new request that will later be sent to the server. The *user* and *password* parameters are optional and only necessary, when the *url* is protected. If *async* is true, the request will be sent asynchronously.

- *onreadystatechange*
  Defines the callback function that will be called, when the asynchronous request has been responded by the server.

- *send(data)*
  Sends the request with the given *data* to the server. If the *data*-parameter is null or omitted, no entity-body (HTTP message body) is transferred to the server. In this case, variables can be sent encoded as query-strings in the *url* of the *open()*-method.

[10] The provided list is only a subset of all available methods and fields. A complete summary of the XMLHttpRequest object can be found at http://www.w3.org/TR/XMLHttpRequest/, last viewed 2008-02-26

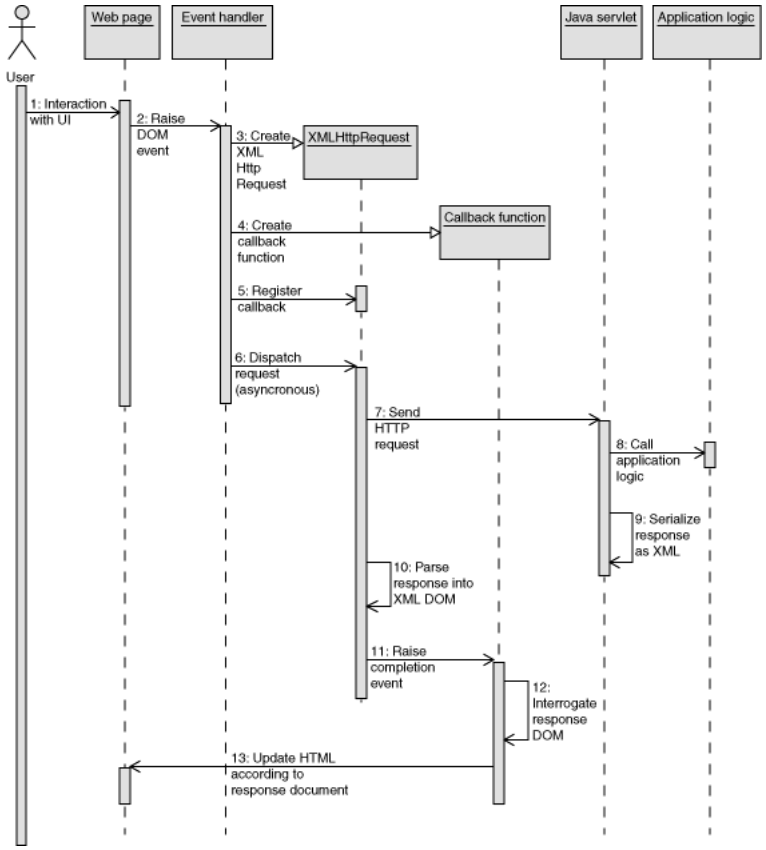


Figure 4.7: The AJAX roundtrip

- *responseText*
  Contains the response the server sends back as a text-string.

- *responseXML*
  If the response returned by the server is XML, this field contains the response value as XML Document Object Model (DOM) tree.

- *readyState*
  Always contains one of the valid ready states of the AJAX call [36,85]:

  - *0*: UNSENT
    Uninialized request, before *open()* is called.
  - *1*: OPENED
    Request has ben assembled (*open()*), but not sent (*send()*).
  - *2*: HEADERS RECEIVED
    Request was sent and is being processed and content headers are available.

  - *3*: LOADING
    Request is being processed and some partial data already is available.
  - *4*: DONE
    The reponse has been completey processed by the server and the data is available to the client.

  Before processing the response data, one should always check the ready state of 4 in the callback function.

- *status*
  Returns the HTTP-status code. Before processing the data, at least status code 200 (OK)[11] should be checked to see if the request was successfully answered by the server.

4.3.2 AJAX for Mobile Devices

As previously mentioned, mobile AJAX means nothing more than the application of the AJAX design pattern on mobile devices and is technologically the same as AJAX on the desktop. Modern mobile browsers use almost the same browser-engines with the same full range of available technologies as their desktop counterparts. The difference rather lies in the constraints and limitations of mobile device hardware and wireless networks, as well as the different handling of a mobile device compared to a a desktop computer.

Some of the unique requirements and characteristics of mobile web applications are [4, 21, 61, 73, 84]:

- **Small Screen Size**
  The screen size of mobile devices is much smaller (somewhere between 120 and 380 pixels) than that of desktop computers. This means that content should be presented linearly without the use of frames, formatting tables or complex pull-down menus.

- **Resource and Execution Limits**
  Mobile devices have very limited memory capacities, which means that caching and displaying big files is not possible. E.g. the iPhone imposes a 10 MB limit on every single downloaded file (HTML page, JavaScript file, CSS file, images, ...). According to Yahoo's UI Research Group [73], the actual size that the iPhone can handle is much smaller and depends on memory fragmentation and other applications that are running concurrently in the browser. Even more limited is the cache of the iPhone: The Yahoo Research Group found out that

[11] A complete list of HTTP status codes can be found at: http://www.w3.org/Protocols/rfc2616/rfc2616-sec6.html, last viewed 2008-02-26

only individual components of less than 25 KB can be cached by the device. The maximum limit of cachable components was determined at 19, resulting in an overall browser-cache of 475 KB to 500 KB.

Another big constraint of mobile devices is the limited processing power. This means that especially calculationally intensive client side scripts will not succeed. Many mobile devices have upper limits for script execution to guarantee the responsiveness of the UI and prevent the battery from draining due to heavy CPU utilization. Again for the iPhone for example, the maximum time for JavaScript execution is limited to 5 seconds for each top-level entry point [4].

- **Limited Key Input and User Interaction**
  Mobile devices follow different paradigms of user interaction than desktop computers. Mobile web applications have to be designed to work under that circumstances. Handsets may e.g. have no full QWERTY-keyboard, making key input tedious. Many new mobile devices are using touch-screens as their major input device. This means that webpages have to be designed in such a way that buttons, links, menus, etc. can easily be captured with the finger. As a consequence this imposes some minimum size on these elements, since pointing with fingers is much less accurate than the traditional mouse available on desktop systems.

- **Limited Network Connection**
  Network transfer speed of mobile devices over GPRS, EDGE, or even HSDPA is much lower than high bandwidth land-based Internet. So its basically a good idea to compress HTTP-responses. The most effective and most widely deployed method is gzip – Approximately 90 % of today's Internet traffic is served by gzip-enabled browsers [61]. Also many mobile browsers like Safari for the iPhone, or Opera Mobile and Mini support gzip. Gzipping can reduce the response size by up to 70 %. The best results are achieved if all textual responses like (X)HTML, JavaScript, CSS, XML or JSON are compressed. Binary files like PDF, or images should not be gzipped since they are already compressed in their own format by default.

  Also mobile web sites are required to be kept especially "tidy". It is wise to reduce the content transferred over the network to an absolute minimum. This can be achieved by using lean HTML-markup (e.g. `<div>` instead of `<table>`) and removing all unneeded payload like HTML-comments or orphaned CSS-classes and JavaScript functions.

AJAX enabled Mobile Browsers

Like for many other mobile application development technologies, device and platform heterogeneity and the resulting fragmentation is one of the biggest concerns for mobile AJAX applications. Mere JavaScript support is not enough for a browser to be AJAX enabled. All of the core technologies making up AJAX (see section 4.3.1) need to be implemented.

The following is a list of the currently most popular mobile AJAX aware browsers. This is by no means a complete list of *all* AJAX enabled browers. Like the mobile device market, the mobile browser market is much less consolidated than that of desktop browsers. Also, the key players in desktop browsing do not find a one to one representation in market share of mobile browsers.

- **WebKit based browsers**
  The Open Source web browser engine WebKit is increasingly becoming popular for mobile devices. It particularly gained fame as the basis for the Safari browser of the iPhone. The project's website can be found at http://webkit.org/.

  Important mobile browsers based on WebKit are:

  - **Safari for iPhone & iPod touch**
    (https://developer.apple.com/iphone/devcenter/)

  - **Nokia S60 web browser**
    (http://opensource.nokia.com/projects/S60browser/)

  - **OHA Android browser**
    (http://code.google.com/android/what-is-android.html)

- **Internet Explorer Mobile**
  (http://www.microsoft.com/windowsmobile/software/iemobile.mspx)
  Like on the desktop, every Windows Mobile device comes with Internet Explorer (IE) pre-installed. Internet Explorer Mobile is a scaled down version of Internet Explorer tailored specifically for wireless mobile devices.

  A good overview of the AJAX capabilities of Internet Explorer Mobile is provided by Kevin Grey, member of the IE Mobile Team, in the IEMobile Team Weblog entry "AJAX on IE Mobile" [19].

- **Opera**
  The Norwegian company Opera currently offers two browsers for mobile devices:

  - **Opera Mobile**
    (http://www.opera.com/products/mobile/)
    Opera Mobile is a full fledged mobile browser with large standards support. It is available for Symbian UIQ and S60, as well as Windows Mobile. In contrast to its desktop brother, Opera Mobile is not free of charge. At the time of writing of the thesis, the unit price was EUR 19,00.
  - **Opera Mini**
    (http://www.operamini.com/)
    As opposed to its near relative Opera Mobile, Opera Mini can not be considered a full AJAX enabled mobile browser. Opera Mini is a proxy based browser that loads every request over a remote server which converts each page into the light-weight Opera Binary Markup Language (OBML) before sending it to the client [38]. The Opera Mini servers are based on the Opera 9.5 engine which does all the processing of the page. The Opera Mini client does no JavaScript processing at all and works more or less as an interface window for an Opera running in the server. JavaScript events are restricted to the most basic ones that are triggered by a user click (`onChange`, `onClick`, links, form-submits, etc.) and enforce a changed version of the page being rendered by the server. Although the XMLHttpRequest object is supported, many AJAX applications won't work as expected on Opera Mini [38]:

    > *Given handset limitations and Opera Mini's client-server architecture, "Ajax" applications cannot be expected to work as expected on Opera Mini.*

    Opera Mini can be downloaded free of charge and since it is written in Java ME, it is available for a broad number of mobile devices.

- **Mozilla Minimo**
  (http://www.mozilla.org/projects/minimo/)
  Minimo is the mobile device browser from Mozilla Foundation. It can be downloaded free of charge, but is only available for Windows Mobile based devices. In October 2007, Mozilla's vice president of engineering Mike Schroepfer, announced that Mozilla Foundation will release a full-featured mobile version of their populator desktop-browser "Firefox", based on the new Gecko engine version 1.9, in 2008 [58]. The Minimo project thus is very likely to be discontinued.

- **ACCESS NetFront**
  (http://www.access-company.com/products/netfrontmobile/browser/index.html)
  The ACESS NetFront browser in its current version 3.5 is a full-featured mobile device browser with rich standards support. It is

available on a broad number of mobile operating systems, including amongst others Palm Garnet/OS, Windows Mobile, Symbian and various embedded Linux distributions. The ACCESS NetFront browser is pre-integrated by handset manufacturers into their devices and is not available as download to end-users.

- **Openwave**
  (http://www.openwave.com/us/products/client_products/mobile_browser/)
  Openwave browsers have been integrated in many mobile phones as the on-board browser. There are currently three different versions of Openwave browsers, of which two of them (Openwave Surfer Browser and Openwave Mercury Browser) fully support AJAX. Like the ACCESS NetFront browser, the Openwave browser is not available as an installable application for the handset user.

4.3.3 Benefits and Limitations

Web based mobile applications offer many advantages over their native or Java ME counterparts. But they also suffer from the same limitations as a desktop browser application, which is most notably the inability of accessing the device capabilities or the platform APIs. This inability weighs particularly heavy on mobile devices as it greatly reduces the information about the ever-changing device context and thus reduces the potential for applications that intelligently adapt to the mobile user on the move. Consequently one of the major findings of the Workshop on mobile AJAX[12], jointly held on 28 September 2007 by the W3C and the OpenAjax Alliance[13], was that mobile AJAX applications need scripting APIs that offer access to device services such as GPS, PIM, messaging, camera, etc.

The following is a short summary of some of the benefits and limitations of mobile AJAX applications:

\+ **Security**
  As a web application, mobile AJAX applications are inherently secure. All scripting code is executed in the browsers sandbox, which only exposes a safe set of APIs to the developer. Additionaly, the XHR object can only make requests to the same domain on which it is running [36], which is referred to as *same origin* policy. For most browsers the same origin policy is very restrictive. E.g. for Mozilla based browsers it means that the protocol, the port and the host need to be identical[14].

[12] http://www.w3.org/2007/06/mobile-ajax/report.html, last viewed 2008-03-06
[13] http://www.openajax.org, last viewed 2008-03-06
[14] http://www.mozilla.org/projects/security/components/same-origin.html, last viewed 2008-03-06

+ **Deployment**
  The deployment is one of the biggest advantages of any web application, which is as easy as changing the files on the server and pressing the browser's reload-button.

+ **Rapid Application Development (RAD)**
  Web applications are generally faster and easier to develop than native applications. With the convergence of mobile web standards into one web, web application developers can greatly benefit from their already acquired skills without the need to learn a new technology. Also many existing web applications only have to be modified to fit mobile devices, which avoids creating a new application from scratch and may serve as a quick entry to mobile computing.

± **Cross platform**
  Web applications are operating system agnostic in the sense that they are only relying on the browser as their development platform. In theory, this allows mobile AJAX applications to be available on a large number of heterogeneous devices, ranging from low-priced feature phones to highend integrated smartphones. As mentioned above (see section 4.3.2) in reality the mobile browser landscape is very fragmented, with only a small number of browsers offering AJAX support. Also the degree of AJAX standards support may vary between different mobile browsers.

– **Access to device capabilities**
  The nonexistent access to the device capabilities is probably the biggest handicap of mobile AJAX compared to other applications. The perceived knowledge about a website's environment is usually referred to as *delivery context*[15]. The term describes a mobile website's ability to access device capabilities like PIM, the local filesystem, position data, etc., as well as information about the network connection or the user preferences. Many efforts are being made at the moment for offering a richer delivery context to mobile AJAX applications [42]:

  – **Browser extensions**
    Next generation mobile browsers could offer direct DOM access to the most important device capabilities, like PIM or position data. Ideally they would be standardized like the members and methods of the XMLHttpRequest object.

  – **Linking of Java and JavaScript**
    The fact that Java ME is already available on a large number of today's mobile devices and that it offers various levels of device

[15] http://www.w3.org/TR/di-gloss/#def-delivery-context-v2, last viewed 2008-04-03

capabilities like File & PIM (JSR 75) or Location (JSR 179) makes it an ideal candidate for connecting to through a mobile AJAX webclient. Java-to-Javascript communication[16] is not new and has been around for a long time to allow Java-Applets embedded in a website to communicate with their environment. Similar techniques could be incorporated in mobile browsers to establish a communication interface between JavaScript and the Java ME APIs available on the device.

- **Native platform extensions**
  Native platform manufacturers could offer proprietary interfaces for accessing parts of their device capabilities within mobile AJAX applications.

Amongst others, the following concrete implementations of mobile browser access to device capabilities have been available at the time of writing of this thesis:

- **Apple iPhone application links [8, Chapter 6]**
  The iPhone application links are similar to the Wireless Telephony Application Interface (WTAI) of WAP introduced in section 4.2.1 in that both allow access to the device via special hyperlinks. The most important difference is that by contrast to WTAI, the iPhone application links can only be used to *open* an application associated with a particular type of hyperlink. They can not be used to *save* any data on the device. The supported special links are:

  mail links, which open the iPhone mail application, e.g.:

```
1 <a href="mailto:frank@wwdcdemo.example.com">John Frank</a>
```

  phone links, which launch the phone application and dial the given number, e.g.:

```
1 <a href="tel:1-408-555-5555">1-408-555-5555</a>
```

  map links, which open a Google map containing the given destination, e.g.:

```
1 <a href="http://maps.google.com/maps?q=cupertino">Cupertino</
      a>
```

  and YouTube links, which open the YouTube application and play the given video, e.g.:

```
1 <a href="http://www.youtube.com/watch?v=video_identifier">
      Some Video</a>
```

[16] http://java.sun.com/j2se/1.4.2/docs/guide/plugin/developer_guide/java_js.html, last viewed 2008-04-03

– **Vodafone MobileScript [47]**
Vodafone MobileScript is an ECMA Script (JavaScript) extension that allows developers to access device APIs and data from within their mobile web applications. It is currently available as download for Windows Mobile 2003 and Windows Mobile 5.0. According to the project website[17], implementations for other platforms such as Symbian or Linux are also planned. Figure 4.8 [47] shows a shematic overview of how MobileScript fits into the architecture of a mobile operating system.

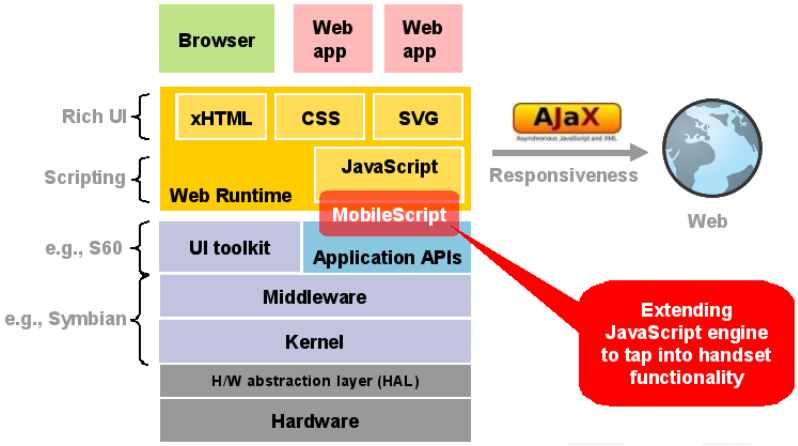


Figure 4.8: Vodafone MobileScript within a mobile operating system stack

The following code snippet shows a simple MobileScript JavaScript function that puts down any incoming call and sends the caller a SMS that the callee is busy:

```
1 function handler() {
2   Phone.IncomingCallRegistry.IncomingCall.PutDownIncoming();
3   Phone.Outbox.SendSMS(Phone.IncomingCallRegistry.
        IncomingCall.SourceAddress, "sorry, busy, I will call
        you later");
4 }
5
6 Phone.OnIncomingCall(handler);
```

– **Opera Platform / Opera Widgets**
The Opera Platform describes a set of JavaScript DOM APIs that allow operators to integrate online content with local device applications [9, 43]. The provided APIs include access to device status flags, messaging, operating system applications and related

[17] http://www.vodafonebetavine.net/web/MobileScripting, last viewed 2008-05-30

features. According to [42], the Opera Platform has been superseded by Opera Widgets[18]. By contrast to the Opera Platform, Opera Widgets are not created by mobile operators but can be made by any developer. They do not support the Opera Platform JavaScript device API interface. A JavaScript File I/O API[19] has been recently announced for Opera Widgets, which allows web applications to interact with the device's file system. Possible file operations include opening, writing, creating, moving and deleting of files and directories.

– **Network dependency**
Browser applications are inherently network dependent and may not even be loaded without access to the network. Also, their possibility to store data locally is very restricted, which makes them impractical for offline use.

[18] http://widgets.opera.com/, last viewed 2008-05-30
[19] http://dev.w3.org/2006/webapi/fileio/fileIO.htm, last viewed 2008-05-30

Chapter 5

Service Oriented Architecture

5.1 Overview

The previous two chapters *Native Development Platforms* and *Mobile Web Applications* showed that one of the biggest problems of mobile application development is the huge fragmentation of operating systems and platforms. There exists no technology that allows for easy cross-platform mobile application development. In such a highly fragmented and frequently changing environment, the only way to address multiple client devices may be to program multiple client applications. In a distributed application scenario, in which the client-code heavily relies on interaction with a server-system, it is desired that the server exposes only one interface to the various clients. Each client can access this interface in a well defined and uniform way, without being aware of the actual implementation of the logic on the server. Also the server is not aware of the clients using its service. It is just a provider of an autonomous interface. This leads to a clear separation of client logic from the server. Such a scenario can be realized with a *Service Oriented Architecture (SOA)*.

5.2 Basic Concepts

The evolution of software packaging spans from functions and methods over packages, objects and classes to components and finally *services* [81].

A *service* in a programming-context is very much like a component. They are both independent building blocks that collectively represent an application environment [17]. But services have several characteristics that set them apart from components. The most important one being *autonomy* [17]. Autonomy means that a service can completely function on its own without being integrated into a larger system. Also services are completely independent from other services. By providing well-defined interfaces, services can be coupled together into greater logical units. This is commonly referred to

as *loosely coupling/bounding* because the services do not have to be aware of each others functionality. This allows for the distribution of services over system, platform and network borders. The services rely on the interfaces for interacting with each other. Two major aspects within this approach are the uniformely accessible interfaces and a standardized way of exchanging information over them.

Another aspect of a service is that it breaks with the traditional client-server pardigm, in which one side is the selected server and the other side the selected client. As a self-contained component, a service can function as a server as well as a client. This means that a service A could act in one situation as a server to service B requesting something and in another situation B could act as a server for A, which itself demands a certain functionality.

In a SOA-architecture, service providers usually register their services in a public registry. Service consumers can query this repository to find services matching their specific criteria. If the registry can offer such a service, it provides the consumer with an endpoint-address where the service can be found as well as a contract describing the service. Figure 5.1 illustrates this *find-bind-execute* cycle [30].

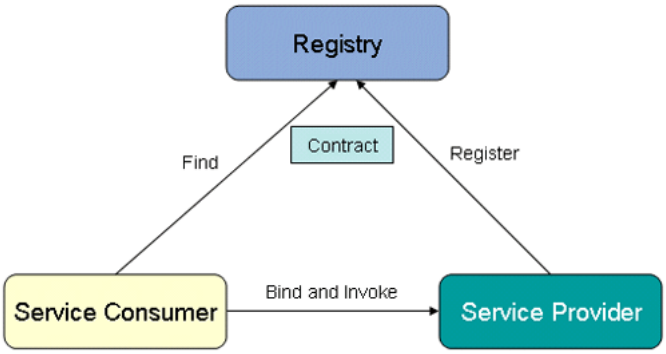


Figure 5.1: SOA find-bind-execute cycle

5.3 Web Services

A Service Oriented Architecture can be achieved in many different ways, but *Web Services* are definitely the most popular one. In fact they are so common that the two terms are often used interchangeably.

Web Services are a realization of the SOA paradigm with existing Internet and WWW technologies, foremost the Hypertext Transfer Protocol (HTTP) and the eXtensible Markup Language (XML) [18]. HTTP is used as the transport mechanism for XML based service message exchanges. Web

Services are not exclusively bound to HTTP yet its widespread deployment in existing Internet environments makes it the primary transport protocol for today's Web Services.

Two important architectural concepts form the Web Services application stack. The more formal *SOAP* approach that builds on a large set of standards defined by the W3C, OASIS[1] and other standard bodies. And the less specified *REST* approach.

5.3.1 SOAP based Web Services

The three core technologies of SOAP based Web Services are Simple Object Access Protocol (SOAP), Web Service Description Language (WSDL) and Universal Description, Discovery and Integration (UDDI). Together they are used as a direct application of the SOA find-bind-execute cycle (see figure 5.1). The *Service Provider* must describe its SOAP based Web Service with WSDL (low level) and UDDI (high level for lookup). A *Service Consumer* can look for a specific service in an UDDI-*Registry*. If the Service Consumer finds an appropriate service, it can bind to it with the information provided by the registry and invoke it via its WSDL service description. The actual data exchange between the service endpoints is carried out with SOAP.

SOAP[2] and WSDL[3] are standardized by the W3C, whereas UDDI[4] is a standard maintaned by OASIS.

SOAP

SOAP (originally an acronym for *Simple Object Access Procotol*, now it is just a name [40]) is the key technology for SOAP based Web Services and responsible for exchanging data between to service endpoints. Like its companion technologies WSDL and UDDI it is built on top of XML and provides a schema in the form of the SOAP-specification that allows for system independent data exchange.

It has been standardized in two different versions, SOAP 1.1[5] and SOAP 1.2[6].

A SOAP document is referred to as a *SOAP message* or *SOAP envelope* [40]. Each SOAP envelope consits of a *header* and a *body*. The header carries meta-information, quality of service constraints and routing directives relevant for the service exchange. The body contains the payload in form

[1] Organization for the Advancement of Structured Information Standards (http://www.oasis-open.org), last viewed 2008-03-11

[2] http://www.w3.org/TR/soap/, last viewed 2008-03-20

[3] http://www.w3.org/TR/wsdland, last viewed 2008-03-20

[4] http://uddi.org/pubs, last viewed 2008-03-20

[5] http://www.w3.org/TR/2000/NOTE-SOAP-20000508/, last viewed 2008-03-20

[6] http://www.w3.org/TR/soap12-part1/, last viewed 2008-03-20

of a (partial) XML document. Binary data can be included encoded in the message body.

SOAP can be used for Remote Procedure Calls (RPC) as well as literal document exchange. For RPC, the body contains XML serialized method calls, which are answered by the remote station with a SOAP message containing the return values. This way Web Services can be used as an alternative to programming language specific RPC methods like amongst others Java RMI, CORBA or Microsoft DCOM.

Today much more common than RPC is literal document exchange. With literal document exchange, XML serialized documents are transferred in the SOAP body. This could be e.g. a booking request containing all the relevant booking data generated by the calling Web Service, which gets answered by the called service with a confirmation or an error message.

An important concept of the SOAP messaging framework is the SOAP *processing model*. Every SOAP message originates at an initial SOAP sender and reaches a specific ultimate receiver via zero or more SOAP intermediaries[7]. The way a SOAP message travels in a network is described as its *message path* [81]. The message path information is included in the SOAP header.

Web Services Description Language (WSDL)

Together with SOAP, WSDL forms the heart of every SOAP based service architecture. It is used for describing Web Services with XML based metadata information in such a way that other services can automatically use them. This way it is a key enabler for a Service Oriented Architecture, as it allows for programmatically exploring and using a service. The currently most widely deployed version is WSDL 1.1[8]

Each WSDL document consists of a set of *definitions* (`<definitions>`-root-tag) that describe the *what*, the *how* and the *where* of a specific service. Listing 5.1, taken from the Mindbreeze Enterprise Search SDK [39], shows an exemplary excerpt of a WSDL-document describing a query-interface (full description of XML Schema messages has been omitted). In detail, a WSDL document comprises the following information [40, 81]:

- **Accurate description of the exchanged messages (*what*)**
  In order for the service consumer to know how to invoke the service and what the corresponding response looks like, the messages exchanged by a service have to be precisely described by the service provider. Usually XML Schema is used for that XML data description. Since WSDL imposes no restriction on the data declaration language, other schema languages like RelaxNG or Document Type Definitions (DTDs) may

[7] http://www.w3.org/TR/soap12-part1/, last viewed 2008-03-20
[8] http://www.w3.org/TR/wsdl, last viewed 2008-03-20

also be used. The schema information of the exchanged messages is located in the `<types>`-tag. The description of the message is in the `<message>`-tag. The `<portType>`-tag groups a related set of messages into one operation. It also defines the kind of the operation which can be either

- *One-way*
  The service only receives a message and doesn't respond.
- *Request-response*
  The service receives a request message and produces a correspoding response.
- *Solicit-response*
  The inverse of request-response: The service firstly sends a message and than receives a response.
- *Notification*
  The service sends a message without a response.

The kind of operation is constructed by arranging the messages that make up the operation with successive `<input>`- and `<output>`-tags.(E.g. only an `<input>`-tag for a one-way operation, an `<input>`-tag followed by an `<output>`-tag for request-response, etc.)

- **How to format the messages (*how*)**
  The information about how the exchanged messages should be bound to a particular messaging protocol is defined in the `<binding>`-tag. The most important messaging protocol is SOAP, which offers the binding types *RPC/encoded*, *RPC/literal*, *document/encoded* and *document/literal*. The choice of the binding influences how the transferred messages ultimately look like.

- **Where to find the service (*where*)**
  The `<service>`-tag "ties together" the information specified through the other tags in the WSDL document into a cohesive interface and gives the service a name. Furthermore it indicates where the service can be found on the network by providing a public location Uniform Resource Locator (URL).

```
<?xml version="1.0" encoding="utf-8"?>
 <wsdl:definitions xmlns:wsdl="http://schemas.xmlsoap.org/wsdl/"
  xmlns:soap12="http://schemas.xmlsoap.org/wsdl/soap12/"
  xmlns:xsd="http://www.w3.org/2001/XMLSchema"
  xmlns:mes="urn:mindbreeze:enterprisesearch:interfaces"
  targetNamespace="urn:mindbreeze:enterprisesearch:interfaces">
  <wsdl:types>
  <xsd:schema targetNamespace="
      urn:mindbreeze:enterprisesearch:interfaces" elementFormDefault="
      qualified">
    <xsd:element name="SearchRequest">...</xsd:element>
    ...
    <xsd:element name="SearchResponse" type="mes:SearchResponse"/>
    ...
  </xsd:schema>
  </wsdl:types>
  <wsdl:message name="SearchRequest">
    <wsdl:part name="parameters" element="mes:SearchRequest"/>
  </wsdl:message>
  <wsdl:message name="SearchResponse">
    <wsdl:part name="parameters" element="mes:SearchResponse"/>
  </wsdl:message>
  <wsdl:portType name="QueryServicePort">
    <wsdl:operation name="Search">
      <wsdl:documentation xml:lang="en">
        Submits a new query.
      </wsdl:documentation>
      <wsdl:input message="mes:SearchRequest"/>
      <wsdl:output message="mes:SearchResponse"/>
    </wsdl:operation>
  </wsdl:portType>
  <wsdl:binding name="QueryServiceBinding" type="mes:QueryServicePort">
    <soap12:binding transport="http://schemas.xmlsoap.org/soap/http"/>
    <wsdl:operation name="Search">
      <soap12:operation style="document"/>
      <wsdl:input>
        <soap12:body use="literal"/>
      </wsdl:input>
      <wsdl:output>
        <soap12:body use="literal"/>
      </wsdl:output>
    </wsdl:operation>
  </wsdl:binding>
  <wsdl:service name="QueryService">
    <wsdl:port name="QueryServicePort" binding="mes:QueryServiceBinding"
        >
      <soap12:address location="https://localhost:23300/soap"/>
    </wsdl:port>
  </wsdl:service>

</wsdl:definitions>
```

Listing 5.1: Excerpt of Mindbreeze Query Service WSDL

Universal Description, Discovery and Integration (UDDI)

The UDDI specification provides a description of Web Services and allows them to be listed in an UDDI-registry. Much like in a classified directory, service consumers can look up Web Services in an UDDI-registry and get all the relevant information for connecting to them. An UDDI-registry can be run by a company for providing services to their customers, as well as for internal usage. Microsoft, IBM and SAP used to maintain the *Universal Business Registry*, which served as a broker place for Web Services all over the world, but was discontinued in 2006[9].

Since WSDL serves as the "technical instruction manual" for a SOAP based Web Service, UDDI needs to tightly integrate with it. This is done via mappings in the UDDI-document that directly map WSDL entities to UDDI entities [81]. Additionally to the technical mapping of the service, the UDDI document provides meta-information that describes the service and allows for criteria based discovery.

2nd generation technologies

Since their introduction in the year 2000 [81, Chp. 4.1], SOAP based Web Services have been widely adopted and gained big momentum especially in the realm of enterprise applications. To compensate for shortcomings of the early specifications and to address newly evolved needs, numerous new protocols have been added to the core technologies. Amongst others they cover issues such as [81]:

- **Transactions**
  WS-Transaction, WS-AtomicTransaction, ...
- **Security**
  WS-Security, WS-Trust, WS-Privacy, WS-Authorization, ...
- **Quality of Service**
  WS-Reliable Messaging, WS-Policy, ...
- **Service Composition**
  WS-BPEL, ...

The new technologies are not a replacement of the well established specifications SOAP, WSDL and UDDI, which still remain the pillars of SOAP based Web Services. The specifications are often developed jointly by a small group of key vendors like Microsoft or IBM and may later get adopted by standards bodies such as the W3C or OASIS. Because of their business origin, some specifications are competing and at the best become *de facto* rather than *de jure* standards.

[9] http://webservices.sys-con.com/read/164624.htm, last viewed 2008-03-22

5.3.2 REST based Web Services

The idea of Web Services based on the REpresentational State Transfer (REST)[10] principle is to directly rely on the technologies already provided by the World Wide Web, rather than creating new ones.

The key technologies for REST are [48]:

- Uniform Resource Identifiers (URIs)
- HTTP
- XML

While SOAP-based Web Services also use these specifications as the *fundament* for higher level protocols, REST *directly* relies on them for providing a Service Oriented Architecture. In contrast to SOAP based services, which reduce the *application layer* HTTP protocol to a *transport protocol* that could easily be substitued by any other protocol for transportation like e.g. SMTP, XMPP or JMS [74], REST Web Services directly leverage the methods provided by HTTP for CRUD (Create, Read, Update, Delete) operations. In the REST world the HTTP methods are called *verbs*, which the developer can use to describe the intended action. The relationship between CRUD-operations and HTTP methods is shown in table 5.1 [75].

General Action	HTTP Method
Create	PUT
Read	GET
Update	POST
Delete	DELETE

Table 5.1: Relationship between HTTP methods and CRUD operations

The proper use of these HTTP methods (verbs) provides RESTful Web Services with a *uniform interface*, as they specify the allowed operations which always leave the service in a consistent state [20,56].

Key properties of RESTful Web Services are [20, 75]:

- **Stateless**
  Rest based Web Services are stateless, which means that each request travelling from client to server must always contain all the necessary

[10] The term REST was coined by Roy Thomas Fielding in his PhD thesis "Architectural Styles and the Design of Network-based Software Architectures" http://www.ics.uci.edu/~fielding/pubs/dissertation/top.htm, last viewed 2008-03-24

context to understand the request. A good test whether a Web Service should be designed RESTful could be to test if the interaction with the service survives a restart of the host without any impact.

- **Resource identification with URIs**
  RESTful Web Services are built with URIs that uniquely identify resources. The service provider publishes a service under an unique URI which can be accessed by the service consumer over HTTP by using the appropriate verbs.

- **Communication through the transfer of representations of resources**
  The response of a request is the *representation* of a REST Web Service resource and is usually transferred in the form of *pure* XML, although any other format like e.g. JavaScript Object Notation (JSON) is also imaginable.

As a must in any Service Oriented Architecture, RESTful Web Services too have to be described in order to be properly invoked. Opposed to SOAP based Web Services which precisely describe the exchanged messages in a WSDL-file, there exists no such specification for RESTful services. But WSDL can be "borrowed" from SOAP services – Instead of the traditional SOAP-binding, the newly introduced HTTP-binding of WSDL 2.0 could be used to describe the service on the basis of HTTP [14].

Another alternative is to describe the invocation in an humanly understandable format and make it publicly available (e.g. in the form of an HTML-page, or a WSDL-file that is read by a human designing the service consumer) [16].

5.4 Mobile SOA

Like AJAX for mobile devices (see section 4.3.2), *mobile SOA* is not a new paradigm but the application of an existing approach in the realm of mobile computing. Basically any Service Oriented Architecture can be applied on a mobile infrastructure, as long as the device (platform) supports it. In contrast to proprietary solutions like DCOM, .Net Remoting, CORBA, Java RMI, etc., Web Services offer the compelling advantage of being platform and programming language agnostic. This is espescially important for the fragmented mobile device market (see chapter 3), in which it is by no means unlikely that the service producer residing on the server is programmed in another language than the consuming mobile client.

Another advantage of Web Services is that they levarage standardised web technologies like HTTP, which transports traffic over well-defined ports that should not normally be blocked by a firewall.

5.4.1 SOAP-based versus RESTful Web Services

Given their limited nature, mobile Web Services are virtually exclusively service consumers. In fact almost all mobile Web Services frameworks like Java ME Web Services Specification (JSR-172) or the .Net Compact Framework Web Services stack exclusively allow the consumption of Web Services [44, 83]. This means that the mobile Web Service developer may not have the possibility to choose between SOAP and REST but has to stick to whatever interface the server offers. If the server architecture can be influenced, or the service producer offers both a RESTful and a SOAP-based interface, the decision has to be made which approach to incorporate.

The following list compares SOAP-based to RESTful Web Services in the context of mobile computing:

- **Requirements**
  RESTful Web Service consumers are technologically less demanding than their SOAP-based counterparts. The minimum requirements for both approaches is an HTTP-connection and a XML-parser, as well as optional threading-support for augmenting the responsiveness by creating the service call non-blocking. SOAP services also require a SOAP API (also SOAP stack) for automatic message creation and unmarshalling according to the provided WSDL-file.

  Many of the native platforms described in chapter 3 offer a SOAP stack. Alternatively Java provides a SOAP API in the form of JSR-172 and the Open Source kSOAP library. All implementations in common is that they only support a subset of the technologies available to desktop SOAP stacks [57].

- **Network traffic**
  In RESTful Web Services, only the core XML message-body is transferred over the network. No SOAP-headers or additional layers of SOAP-elements need to be communicated [75]:

  > *REST is particularly useful for limited-profile devices such as PDAs and mobile phones, for which the overhead of headers and additional layers of SOAP elements on the XML payload must be restricted.*

  This limits the overall amount of data needed for communication, which can be a great performance benefit for resource scarce limited wireless networks.

  Opposed to a SOAP-request message-envelope, RESTful Web Service requests can be made using the HTTP-GET method with a few query-string characters as parameters.

- **Response parsing**
  Concerning response parsing both approaches are equal. The mere SOAP-message body without the SOAP-headers is usually equivalent to an REST-XML message, which means that both have to be processed in an appropriate way by the consuming service.

- **Processing model**
  The processing model is one of the strenghts of SOAP. It describes how headers have to be processed by intermediary nodes on receiving a SOAP message. This way it can influence the message path by enforcing qualities of service like encryption or reliable and acknowledged delivery [81]. This is a feature completely missing out of the box in REST-based Web Services and would have to be built manually.

- **Error handling**
  Fault handling is another very useful feature of SOAP-based Web Services. The five fault codes (VersionMismatch, MustUnderstand, DataEncodingUnknown, Sender, Receiver [81]) can be used for automatic error handling. Additionally SOAP fault messages contain human readable error-description, which can help in building and debugging the software. REST-messages do not provide a built-in fault-handling model. A common approach is to silently ignore faulty messages.

In summary RESTful Web Services tend to be more applicable for constrained mobile environments than SOAP-based services. The lower technical requirements and the overall lower network traffic should usually outweigh the benefits of a strong processing model and the well defined error handling offered by SOAP Web Services.

Even the creators of the Open Source kSOAP Web Service client library for constrained Java environments advice their users to use RESTful Web Services for mobile devices whenever possible[11]:

> *Please note that SOAP introduces some significant overhead for web services that may be problematic for mobile devices. If you have full control over the client and the server, a REST based architecture may be more adequate.*

5.4.2 Security

The security issue will not thoroughly be expanded on in this paper, but at least the most important mechanisms and possibilities for securing a mobile Web Services application should be mentioned.

[11] http://ksoap2.sourceforge.net/, last viewed 2008-03-27

First of all, the chosen platform must at least have support for Secure Socket Layers (SSL) or Transport Layer Security (TLS) in order to provide basic encryption and authentication mechanisms. This is by no means self-evident for mobile devices, as cryptographic ciphers and secure network protocols tend to be very CPU intensive and thus can quickly drain the device's battery or even be incomputable for very weak CPUs. As described in section 3.2, SSL and TLS support was not available in the Java ME MIDP until version 2. All other platforms introduced in chapter 3, including Java MIDP 2, do of course offer at least basic encryption ciphers and security protocols.

No matter if a mobile Web Service is implemented with SOAP or REST, it will almost definitely be implemented based on HTTP. Such an HTTP based distributed scenario basically offers the following three authentication schemes, which can also be implemented on a mobile device:

- The client holds a trusted certificate that it uses to authenticate with the server over a Public Key Cryptographic System (PKCS) on transport layer.
- The server ensures the authentication via standard HTTP authentication mechanisms.
- The client and the server negotiate an authentication mechanism on top of an application protocol, which encodes the authentication data into the transferred payload.

Chapter 6

Mobile Search Client Prototypes

6.1 Overview

After the first theoretical part that provided an overview about mobile computing, mobile operating systems, mobile web applications and mobile Web Services, this chapter will describe two concrete mobile enterprise search client prototype applications, developed for Mindbreeze Software GmbH.

Many of the findings from the previous chapters directly influenced the presented prototypes or aided in narrowing the problem domain. The prototypes described in this chapter have an explorative character and are meant to showcase the possibilities of a SOA based search client by means of two quite distinct platforms. The first client is a mobile AJAX web application tailored for the iPhone and the second prototype is a native client realized with the Open Handset Alliance's new Android platform.

6.2 Choice of platform

As chapter 3 about mobile phone operating systems showed, there exists a large number of native smartphone application development platforms. In addition, the fragmentation of middleware application platforms like Java is substantially higher than for desktop computing. Many operating systems support Java ME (see chapter 3), yet the available components differ significantly from platform to platform and an on-the-fly installation of missing packages mostly is impossible due to either device, network or operating system limitations. Even web applications that are today developed rather uniformely for desktop computers, due to the browser consolidation and the dominance of Microsoft Internet Explorer and Mozilla Firefox, are difficult to write in an integrative manner for mobile devices. Chapter 4 shows that one has to consider many different browsers that might interpret standard

technologies differently and also the supported technologies vary significantly from static WAP 1.0 to AJAX. Additionally mobile devices have very different form factors with considerably varying screen sizes and input facilities, which further complicates application development.

A simple choice of the platform for a mobile application can consequently not be made and depends on various factors. Amongst others they comprise:

- **Diffusion of the platform**
  One key aspect of a mobile phone application is to reach as many people as possible, though picking the universal platform is virtually impossible. The mobile device market is much less consolidated than the personal computer market with its Microsoft Windows dominance. Also it is subject to rapid changes not imaginable in the desktop market. E.g. the iPhone which has been introduced in summer 2007 already held a market share of 7 % of the worldwide smartphone sales in quarter 4 of 2007 (see table 3.1 on page 14). More information about the most important smartphone platforms is provided in chapter 3.

- **Technical possibilities**
  Different platforms offer different levels of interaction with the device. Usually low level access is required to make use of the device context or to tweak maximum performance. But using low level APIs also means more commitment to a specific platform and usually also involves writing more complex and time-consuming applications. Figure 6.1 depicts the position of the three prevalent mobile development techniques *native code*, *Java ME* and *web applications* in relation to *ease of development* and *functionality and performance* (According to [22]).

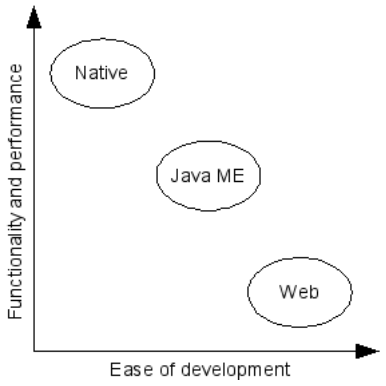


Figure 6.1: Position of native code, Java ME and mobile web applications in relation to ease of development, functionality and performance

- **Organizational constraints**
  Besides the technical considerations there may be various organizational reasons that influence the choice of a particular platform, like amongst others:
  - What devices do my customers have?
  - What development skills does my team have?
  - What platform offers the best deployment scenarios?
  - What platform best fits our existing IT-infrastructure?
  - Which platform has very good future prospects?
  - Pricing and licensing?

Two very distinct platforms have been chosen as the basis of the prototype enterprise search client applications presented in this paper:

- **Mobile web application customized for the iPhone**
  The Webkit based iPhone browser offers one of today's most sophisticated mobile browsers (see section 4.3.2 on page 69). It provides almost the same set of functionalities as a desktop browser and allows for creating rich web applications with full AJAX support. Due to the use of standard Internet technologies, mobile applications can be developed quite rapidly. The complete lack of device access means that no interaction with the system APIs is possible. Hence the client is restricted to just being a consumer of the search services offered by the server.

  The choice of the iPhone client is mostly motivated by the new evolutionary possibilities it offers in the field of mobile web applications and the high business relevance it already gained since its market launch.

- **Google Android native application**
  The Android client is on the opposite side of the technological spectrum, as it represents a native development platform with full access to the device APIs. It provides e.g. access to positioning data, PIM (contact lists, calendar data, etc.), call logs, mass storage cards (SD-cards), etc. While any native development platform described in chapter 3 of this paper could have been elected as the native client platform, Android was chosen since it provides a new application model tailored for mobile applications, which could potentially enable the creation of new up to now unexpected mobile applications. As one of the newest mobile smartphone platforms, Android shows quite well which direction mobile computing could take in the very near future. Also Google's strong market position could make it an economically important platform in the future.

The goal of the prototypes was not their direct mass marketability, but to show and evaluate what can already be done today with the respective technology as the basis for a mobile enterprise search client. If instant marketability was the prevalent criterion, the choice of platforms would have been different. According to diffusion of the platform, possible candidates could have been e.g. a WAP 2.0 web client and a S60 or UIQ (Symbian OS) native client. Another promising candidate could have been Java ME, which technologically sits in between these two paradigms and has a large (though fragmented) diffusion accross various devices.

6.3 General Setup

The basic setup is the same for both clients: A Web Services consumer that issues search queries to the Mindbreeze Enterprise Search (MES) server and gets a search response back as result. The MES provides a SOAP based as well as a RESTful Query Service for querying information from the server. On a standard installation, the SOAP Query Service can be reached under the URL `https://localhost:23300/soap` and the REST Query Service under `https://localhost:23300/find`. Of course in a real world environment, protocol (http or https), server name and port can be chosen by the system administrator according to the company's needs.

The SOAP API for Java and .Net, distributed with the Mindbreeze Enterprise Search SDK, runs underneath a set of high level wrapper classes, which abstracts the use of SOAP from the application developer.

6.3.1 RESTful Web Service

Out of various reasons explained in more detail in section 5.4 about mobile SOA, the REST based Query Service has been chosen as the communication interface to the search server. It allows for querying results via GET as well as POST. The GET-request is formulated with a set of specified query-parameters whereas the POST-request is made with an XML document as payload. The GET- and the POST-method have exactly the same power, which means that every GET-request can always be formulated as a POST-request and vice versa.

The returned response always is in the form of an XML document. The XML-Schema for creating the request (SearchRequest), as well as the schema defining the XML response are precisely documented in the WSDL-file describing the SOAP Query Service in the MES SDK documentation [39, 5.1.2 Query Service]. In fact the exchanged XML messages are exactly the ones contained in the SOAP-body of the SOAP Web Service. This way the WSDL document also serves as a perfect description of the RESTful interfaces.

Listing 6.1 and 6.2 both show an exemplary search request for the term

"Mindbreeze". Listing 6.1 represents the GET-version and listing 6.2 its POST equivalent. The GET variant has the benefits of more precisely abiding by the REST definition[1] and being less verbose which results in less network traffic. The downsides of GET over POST are that for browser applications there is a limit to the maximum length of a GET querystring and that the search terms need to be encoded more precisely.

```
query=Mindbreeze&start=0&count=1&algorithm=default&samplelength=0&
    metadatasamplelength=0&detailedcount=auto&embed-menu=true&embed-icon
    =true
```

Listing 6.1: Instance of a MES Query Service GET search request

```
<SearchRequest sessionid="acf3d86b-95d8-4517-a33f-02884faa6dd6">
  <Query>
    <AndConstraint>
      <StringConstraint language="human" string="Mindbreeze"/>
    </AndConstraint>
  </Query>
  <Options>
    <Range start="0" count="1"/>
    <ContentSamples length="0"/>
    <MetadataSamples length="0"/>
    <Subtotals timeperiod="auto"/>
    <Capabilities>
      <Capability value="tag:mindbreeze.com,2007/contextitems/contextmenu
          "/>
      <Capability value="tag:mindbreeze.com,2007/contextitems/contexticon
          "/>
    </Capabilities>
  </Options>
</SearchRequest>
```

Listing 6.2: Instance of a MES Query Service POST search request

Listing 6.3 shows an exemplary response to the search requests from listing 6.1 and 6.2 (response has been trimmed for clarity).

[1] It overloads the RESTful meaning *update* of the POST-verb with *read* from GET (see section 5.3.2 for more details).

```
<SearchResult>
  <Context>
    <ContextItems>
      <Item id="tag:mindbreeze.com,2007/contextitems/contextmenu;89882
          bf36df7be60" type="tag:mindbreeze.com,2007/contextitems/
          contextmenu">
        <Action name="Open" pattern="file:///$key$"/>
      </Item>
    </ContextItems>
    <MetadataRowsets>...</MetadataRowsets>
  </Context>
  <Hits totalcount="2758">
    <DetailedCount type="discrete">
      <Count timespan="2006">1</Count>
      <Count timespan="2007">10</Count>
      <Count timespan="2008">1</Count>
    </DetailedCount>
    <Hit id="0" ctxmetadatarefid="11527808229362188050" categoryid="File
        " categoryinstance="\\Mestest\C_Share" date="20071023144952" key
        ="\\Mestest\C_Share\Program Files\Mindbreeze\Enterprise Search\
        Server\Additions\White Paper for Mindbreeze Enterprise Search (
        en).pdf" score="56.7164" count="107" title="White Paper for
        Mindbreeze Enterprise Search (en).pdf" categoryclass="pdf" size=
        "1693678">
      <ContextItems>
        <Item type="tag:mindbreeze.com,2007/contextitems/contextmenu" ref
            ="89882bf36df7be60"/>
      </ContextItems>
      <OptDatas>
        <OptData optdatakey="extension" optdatavalue="pdf"/>
        <OptData optdatakey="directory" optdatavalue="\\Mestest\C_Share\
            Program Files\Mindbreeze\Enterprise Search\Server\Additions"/
            >
      </OptDatas>
    </Hit>
  </Hits>
</SearchResult>
```

Listing 6.3: Instance of a MES Query Service search response

Generally the exchanged data is self-explanatory. The number of returned hits can be limited with the `start` and `count` attributes. This is especially important for a mobile environment, as the transferred payload should be limited. Another important feature is that every hit holds a reference to a context menu describing the actions that are possible for the particular content type. For that purpose the `key` attribute of the `<Hit>`-tag has to be replaced with the `$key$` placeholder of the `<Action>`-tag of the corresponding context menu. In listing 6.3 the hit's key is a pointer to a file on the local network. In a real world scenario, the context menu greatly depends on the applications and data types the MES is fed with and what actions each of the applications offers for a particular file.

6.3.2 Security and Authentication

The Mindbreeze Enterprise Search provides many ways of auhentication (see [39, 1.3.5 Authentication and Authorization]) like Kerberos or NT Lan Manager (NTLM). Still, for the prototypes test environment the option "Unrestricted Public Access" was chosen. This grants access to all files of the search index without any kind of authentication. While this would be inappropriate for a real world scenario, it greatly helped to reduce the complexity of the protoype applications and allowed to focus on the mobile peculiarities of the respective client platform.

6.4 Mobile AJAX – iPhone

The purpose of the iPhone prototype is to consume data provided by the Mindbreeze Query Service. It can be seen as the mobile counterpart to the already existing Mindbreeze Search Web Client. It provides a search interface with basic refinement options. Being a mobile web application, it can not make use of the device context or access any data stored locally on the phone.

The iPhone prototype was used as the mobile frontend showcasing the flexibility and power of Mindbreeze Enterprise Search together with Fabasoft Folio, which can e.g. transform non iPhone files on the fly into PDF documents, at the Gartner Portals, Content and Collaboration Summit (PCC) in Baltimore (USA) from March 26 to March 28 2008[2].

6.4.1 Graphical User Interface (GUI)

The GUI of the mobile AJAX client is tailor-made for the iPhone, making it look almost like a native application. As figures 6.3, 6.4 and 6.5 show, at first glance the browser based client is indistinguishable from a built-in application. This is rooted in the fact that the prototype was developed using typical iPhone colors and GUI components and that the URL text field (see figure 6.2 taken from [8]) has been hidden. As shown by comparison with the standard Safari view on the iPhone depicted in figure 6.2, the only difference that clearly identifies the client as a browser application is the button bar at the bottom. By contrast to the URL text field, the button bar can not be hidden/removed programmatically and will always remain visible in every browser application.

The iPhone client was designed with the goal in mind to be as compact as possible and to spare as much navigation from the user as possible. Hence the entire application fits into one single page. As figure 6.3 (a) shows, initially the main query field as well as all refinement options are presented

[2] http://www.mindbreeze.com/mindbreeze/news---events/events/past-events---archive/event_review/gartner_pcc_08.htm, last viewed 2008-04-16

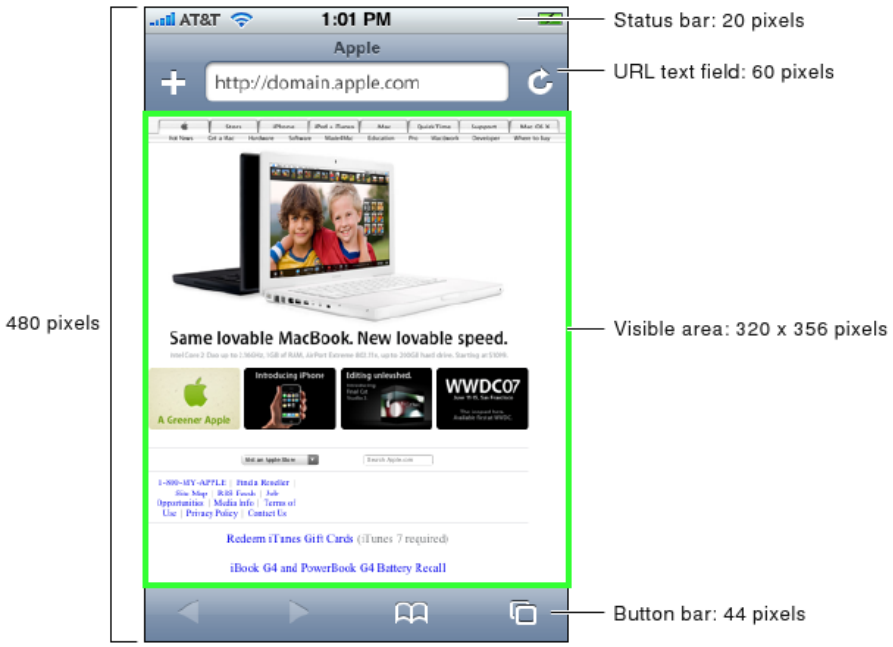


Figure 6.2: iPhone Safari browser standard view

to the user. When the user tips the finger into the query text box the virtual keyboard of the iPhone automatically appears on the screen (see figure 6.3 (b)). The search is initiated as soon as the user presses the *done/fertig* button on the virtual keyboard.

Pushing the *done/fertig* button causes the virtual keyboard to vanish and the search client to fetch the search response from the Query Service. While the search is being processed, the refinements part of the initial search panel slides into the background, making place for the search result list and an optional navigation panel (see figure 6.4 (a)). As indicated in figure 6.4 (a), the only refinement option that stays visible is the *date* field. This has been made to preserve resemblance with the desktop web client, which offers the possibility of quickly refining a search result set by date.

By tipping the finger on an entry in the result list, it gets highlighted and shows its context menu. In the screenshot presented in figure 6.4 (b), the standard "Open" context menu entry is listed. In a real world scenario the context menu usually is much more comprehensive and could offer all kinds of actions associated with a document and the user's context.

Finally, figure 6.5 shows that the Mindbreeze iPhone client seamlessly adjusts its layout if the phone is rotated into landscape orientation.

(a) (b)

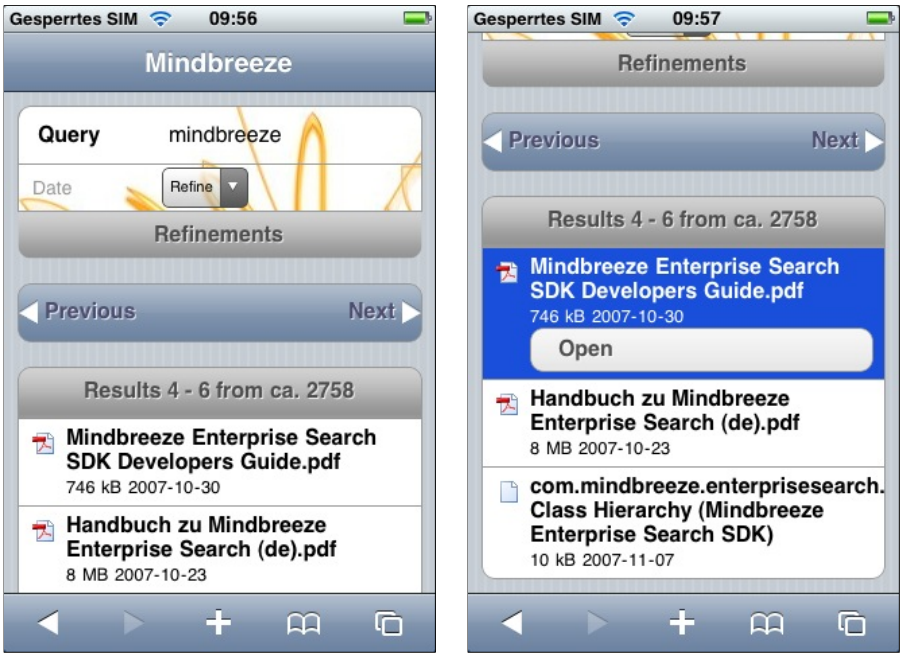
Figure 6.3: iPhone client directly after launching (a) and while typing a query into the query text box with the virtual keyboard (b)

6.4.2 Realization

In order for the iPhone web client to get the *look and feel* of a native application, it uses a small graphical library called iUI[3] developed by Joe Hewitt. The iUI library consists of some image files for iPhone style GUI elements (buttons, arrow, backgrounds, ...), a CSS document and a JavaScript file. Although iUI provides some advanced features like a native looking page transition, with the next page sliding in from the left, the prototype only uses its pinstripe background image and the bluish color gradient for the title and navigation bar (see figures 6.3 - 6.5).

The heart of the application are the methods that handle the AJAX call to the search server and parse the returned result. As a concession to the limited capacity of a mobile device, the prototype does not leverage an AJAX-framework, but provides the AJAX functionality from scratch. This allows for writing more specific code which results in less voluminous JavaScript files. Listing 6.4 shows an excerpt of the function that constructs the `SearchRequest` object and passes the data to the XMLHttpRequest object.

[3] http://code.google.com/p/iui/, last viewed 2008-04-16



(a) (b)

Figure 6.4: IPhone client after the search request has been processed (a) and showing the context menu of a selected entry in the result list (b)

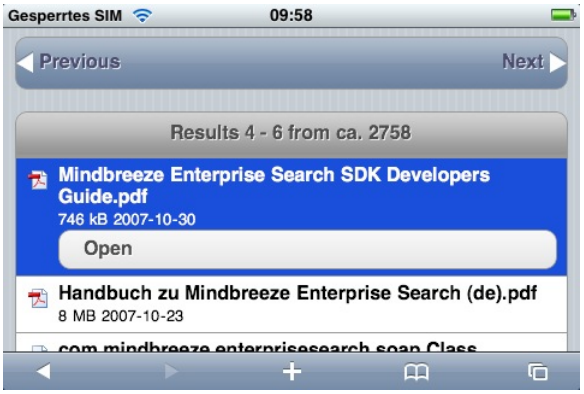


Figure 6.5: IPhone client in landscape mode

```
1 var xhr = new XMLHttpRequest();
2 function invokeSearchWS() {
3   ...
4   var sr = new SearchRequest("acf3d86b-95d8-4517-a33f-02884faa6dd6",
        start);
```

```
5   xhr.onreadystatechange = populateResultList;
6   xhr.open("GET", searchServiceURI + "?" + sr.toHTTPQSString(), true);
7   xhr.send(null);
8
9   // alternative POST-request
10  //xhr.open("POST", searchServiceURI, true);
11  //xhr.send(sr.toXMLString(false));
12 }
```

Listing 6.4: iPhone prototype – Excerpt of JavaScript function invoking the Web Service request

The `SearchRequest` object takes a session-id and the start position of the first hit to be displayed as parameters. All other attributes like the query text or any refinement constraints are directly read from the respective input boxes. Basically the `SearchRequest` object is structured like a Java Bean, with private fields that hold data values as well as setters and getters to set, respectively retrieve them. The collected data can then either be transformed into an HTTP querystring, or into a XML document.

JavaScript does not provide built-in methods for writing XML. Listing 6.5 shows the custom function that is used to construct the XML POST search request.

```
1  function XMLElement(name, content) {
2    this._name = name;
3    this._content;
4    if (content == null) {
5      this._content = [];
6    } else {
7      this._content = content;
8    }
9    this._attributes = {};
10   this.setAttribute = function(key, value) {
11     this._attributes[key] = value;
12   };
13
14   /**
15     Add an XMLElement as content of this element
16   */
17   this.addContent = function(elem) {
18     this._content.push(elem);
19   }
20
21   this.toString = function(prettyFormat) {
22     var xmlStr = "<";
23     xmlStr += this._name;
24     for (var attKey in this._attributes) {
25       xmlStr += " " + attKey + "=\"" + this._attributes[attKey] + "\"";
26     }
27     if (this._content.length > 0) {
28       xmlStr += ">";
29       if (typeof(this._content) == "object"){
```

```
30        xmlStr += (prettyFormat) ? "\n" : "";
31        for (var i = 0; i < this._content.length; i++) {
32          xmlStr += this._content[i].toString(prettyFormat);
33        }
34      } else {
35        xmlStr += this._content;
36      }
37      xmlStr += "</" + this._name + ">";
38    } else {
39      xmlStr += "/>";
40    }
41    xmlStr += (prettyFormat) ? "\n" : "";
42    return xmlStr;
43  }
44 }
```

Listing 6.5: iPhone prototype – JavaScript function constructing XML elements

The `onreadystatechange` attribute shown in figure 6.4 takes the name of the method as value that will handle the asynchronous response and display the returned data to the client, after the response has been processed by the server. An highly trimmed version of this method named `populateResultList` is shown in listing 6.6.

```
1  function populateResultList() {
2    if (xhr.readyState == 4) {
3      if (xhr.status == 200) {
4        var xmlDoc = xhr.responseXML;
5        var root = xmlDoc.getElementsByTagName("SearchResult")[0];
6        //Indicates the last tranche – no more results found for query
7        var endofhits = root.getAttribute("trigger") != null &&
8          root.getAttribute("trigger") == "endofhits";
9
10       //collect the data for the context–menus
11       var contextItems = root.getElementsByTagName("Context")[0]
12         .getElementsByTagName("ContextItems")[0]
13         .getElementsByTagName("Item");
14
15       ...
16
17       var hits = root.getElementsByTagName("Hits");
18
19       //exit with no results found
20       if (hits == null || (hits.length == 1
21         && hits[0].getAttribute("totalcount") == "0")) {
22         $("resultList").innerHTML = $("resultCountLabel").innerHTML = "No
                results found.";
23         return;
24       }
25
26       ...
27
```

```
28        //get all the values and display them in the list
29        $("resultList").innerHTML = "";
30        for (var i = 0; i < hit.length; i++) {
31          //fill the hit-Buffer with the information displayed for each hit
32          //(e.g. name, date, size, directory, ...)
33          var hitBuf = "...";
34          $("resultList").innerHTML += hitBuf;
35        }
36      }
37    } else {
38      $("resultList").innerHTML = "Server can't process the request. Error
            -Code: " + xhr.status;
39    }
40  }
```

Listing 6.6: iPhone prototype – Excerpt of JavaScript function handling the asynchronous response and filling the result list

Firstly, the ready-state and the HTTP response code have to be checked to ensure all data has been received correctly (see section 4.3.1 for details). If everything went well, the resulting XML document can be parsed.

The XHR `responseXML` attribute holds the received XML file as *Document Object Model (DOM)* document. A DOM parser holds the entire XML document in memory and allows for parsing it along all its axis. Another very popular XML parsing concept is the *Simple API for XML (SAX)*. Unlike the DOM, SAX does not load the entire XML document into memory, but parses it serially. Whenever a match (e.g. opening tag, attributes, closing tag, ...) occurs, the SAX parser raises an event that can be handled by the developer. Once a certain token has been parsed it is gone, which means that back references and random access are not possible. The advantage of SAX over DOM is that it needs less memory. More information about DOM versus SAX can e.g. be found in the W3C's DOM FAQ[4].

As shown in section 6.5.4, the Android prototype is built using a SAX parser. For the JavaScript client there is no choice since the `responseXML` field already stores the XML as DOM document. The only other alternative would be to use the string result provided by the XHR `responseText` field and to process it by using regular expressions. Not only would this very likely result in worse performance, but also substantially increase the complexity of the parsing process.

6.4.3 Special Considerations for the iPhone

Generally web application development for the iPhone works like for a desktop computer. Only a few things have to be considered additionally:

- **Limited Screen Size**
  The most apparent difference between an iPhone and a desktop com-

[4] http://www.w3.org/DOM/faq.html#SAXandDOM, last viewed 2008-04-16

puter is the limited screen size. Although the iPhone screen measures 320 x 480 pixels in portrait mode [8], the viewport is initially set to a width of 980 pixel. Though this is a reasonable default for most of today's standard web applications, it is not appropriate for iPhone tailored sites. To change the viewport to the actual size of the phone, the iPhone proprietary viewport-meta-tag has to be used:

```
1 <meta name="viewport" content="width=320; initial-scale=1.0;
      maximum-scale=1.0; user-scalable=0;" />
```

- **Event handling & virtual keyboard**
  Safari on the iPhone does not exactly support the same events as the desktop version. Some events (e.g. `onmouseover`) are not supported at all, others may be raised at different points in the JavaScript event handling model.
  In case of the MES client prototype this is especially crucial for the triggering of the search. The search is fired, when the query text box loses focus, which is modelled in JavaScript with the `onblur` event. This event is raised on the iPhone, when the user hits the *done* button on the virtual keyboard, which is exactly the desired behavior.

- **Limited cache size**
  Compared to a desktop browser, the Safari on the iPhone has very limited cache capabilities (see section 4.3.2). Only individual documents with an overall size of less than 25 KB will be cached by the browser[5]. The limit applies to the unzipped version of the file which means that even if a gzip compressed file has less than 25 KB, it will only be stored in the cache, if its uncompressed size is smaller than 25 KB. As a consequence, too big files (e.g. JavaScript libraries) should be at best avoided or at least split up into several smaller files.

6.4.4 Deployment

The easy deployment definitely is one of the biggest advantages of any web application. Updates and changes only have to be performed at one single location – the server. The actual rollout on the client is as easy as pressing the browsers refresh button.

Special caution has to be taken on the *same origin* policy of AJAX applications. As mentioned in section 4.3.3, only scripts that point to the exactly same server from where the AJAX file has been loaded, will be executed. That is why for the development of the iPhone prototype, a simple Java Servlet proxy had to be written. It resided on the same server on which

[5] http://yuiblog.com/blog/2008/02/06/iphone-cacheability/, last viewed 2008-04-16

the prototype application was and simply passed all requests and responses between the search client and the MES Query Service.

For the presentation of the client at the PCC, the code was directly included into the built-in MES client service that also delivers the Mindbreeze Search Web Client, making the proxy redundant.

6.5 Native Application – Android

As mentioned before, compared to the iPhone web application, the native Android client represents the other side of the technological spectrum in regard to access of device functionalities. The Android platform exposes many APIs to the developer that allow to take advantage of the mobile context or low level device features such as amongst others camera, GPS, compass, accelerometer, PIM (contacts, call logs, SMS, e-mail, etc.), mass storage card or 3D graphics.

In addition to offering the same service consumer functionalities as the iPhone web client, the Mindbreeze Android client will also make use of the so called *content providers* offered by the Android platform. The concept of content providers is unique to the Android platform. In a nutshell they represent an interface that lets applications share data among each other in a uniform way. All built-in Android applications, like e.g. addressbook, calendar, SMS, etc. expose their data as content providers. More information about content providers is given in the next section about the Android platform.

The concept of the Mindbreeze Android search agent is to crawl all the data stored in any content providers found on the device and push it into the index on the server via a Web Service interface. This includes the providers known at installation time like the ones of the built-in applications, as well as any providers exposed by third-party applications at runtime of the client, which allows the client to "grow" alongside the device and the applications being installed. A detailed overview of the architecture of the mobile Android client is given in section 6.5.2.

6.5.1 Android Architecture Overview

The Android platform possesses some unique concepts which the prototype application builds-on. It is important to have a general overview of these concepts in order to fully understand the prototype architecture. As at present no official book about Android has yet been released, all herein presented information was taken from the official Android online documentation[6].

[6] http://code.google.com/android/documentation.html, last viewed 2008-05-07

Key structures

The Android platform consists of five building blocks, which can be used to build an application:

- **Activity**
- **Intent & Intent Filter**
- **Intent Receiver**
- **Service**
- **Content Provider**

Every application is a composition of some or even all of these building blocks, yet not all components have to be incorporated in every Android application.

Activity Activity is the main building block of an Android application. It represents a single graphical screen. This means that every screen that is displayed by an application is an activity of its own. A task list application e.g. that offers the user two screens, one with a list overview of all tasks and one with a detail view of a selected item, would hence have two activities. An activity is composed of *views*, which represent individual GUI components like buttons, lists, textfields, etc. In fact all user interfaces that handle screen layout and interaction with the user are derived from the basic class `android.view.View`. This is a concept similar to that of other GUI frameworks. An overview of the most commonly used views can be found in the Android view gallery[7].

Intent & Intent Filter Unlike the other Android building blocks, intents and intent filters are not responsible for directly operating on data. They are rather the key components in the Android event framework (intent resolution framework) to *start* certain operations[8]. In particular they are responsible for starting activities, intent receivers and services.

As their name implies, intents represent the intention of an application or the system to do something or to inform other applications about something. An intent is expressed by defining an *action* that describes what should be done and the corresponding *data*, which defines the set of data the intent operates on. The data the activity can operate on is defined through a unique Uniform Resource Identifier (URI). The actions are described through

[7] http://code.google.com/android/reference/view-gallery.html, last viewed 2008-05-07

[8] The official Android online documentation does not talk about intent & intent filters as building blocks but rather attaches them to activities, services and intent receivers.

predefined system constants, or through new constants specified by a third party application.

For the understanding of the Android intent mechanism it is important to know that the system differentiates between two different types of actions:

- *activity actions* and
- *broadcast actions*

Activity actions are used to navigate from activity to activity (screen to screen). In Android the code to show a new screen is not directly coded in the calling activity. The calling activity rather expresses in an intent what it wants to be done. This allows for a loosely coupling between user actions and the resulting screens. Amongst the most common activity actions are `MAIN_ACTION`, `EDIT_ACTION`, `PICK_ACTION`, `VIEW_ACTION`, `DELETE_ACTION`, etc.

E.g. an application that wants to display a graphical view of the first record of all people in the address book could create an activity with the action `VIEW_ACTION` and the data URI `content://contacts/1`.

On the other side an acvitiy that wants to show up when a record of a person should be displayed has to describe in its intent filter that it is capable of doing so. An intent filter is defined in the applications deployment descriptor (see section 6.5.1).

The process of resolving an intent happens at runtime. When the user triggers an action that starts an intent, the system looks at the intent filters of all installed components and picks the one that best fits the requested criteria.

This clear separation of data and actions performed on the data is one of the strengths of Android. It is theoretically possible to replace any application (including the built-in ones) with a new one, the user thinks is better suited for a particular task. What is important is that all the applications operate on the same set of data. If e.g. the user chooses to install a new mail client that new application will specify in its intent filters that it is capable of handling all the actions the built-in mail client usually handles. As long as the client remains installed, it does all the mail operations on the same set of data the built-in client previously used. Once the user is bored with the new application and either installs another one or resorts to reuse the old one, no data will be lost, because all clients always operate on the same set of data.

Broadcast actions do not start an explicit activity or service. As their name implies they represent broadcast data that is sent either by the Android system or a third party application. Any application that is interested in a particular broadcast can register an appropriate intent receiver in its deployment descriptor, which gets called whenever the broadcast is made. More information on broadcast actions and intent receiver is provided in the next

paragraph about intent receivers.

Besides of the just explained *implicit* intents, Android also supports *explicit* intents. An explicit intent directly specifies an explicit class name of an activity or a service to be called. It is therefore not matched by the system through the intent resolution mechanism. As a consequence, objects called through an explicit intent do not have to provide an intent filter in the applications manifest. Although this means a tight coupling between two components, this is often desired as it reduces the application complexity. Many times the developer doesn't even want an activity or service to be replaced by another one (e.g. login screen).

Intent Receiver In a nutshell, intent receivers can be used for execution of code in response to an external event, like e.g. when the phone rings, an application package has been installed or removed, the device has been booted, etc.

As the name implies, intent receivers are also closely related to intents. One of the main differences between activities and intent receivers is that intent receivers are *extremly short lived* and *solely* used to handle broadcast actions. Another major difference compared to activities is that they do not display a user interface. They start up when they receive a broadcast they are registered for and then perform some work in their `onReceiveIntent()`-method. Once the `onReceiveIntent()`-method has finished, the intent receiver is done. If an intent receiver does not manage to finish its `onReceiveIntent()`-method within 10 seconds, the Android system will regard the receiver as blocked and mark it a candidate to be killed by the process scheduler[9] (for more information about the application life-cycle see section 6.5.1). If an intent receiver is used to trigger a longer running action (like e.g. network or I/O operation), the application programmer ought to start a service (and possibly spawn a new thread) in the `onReceiveIntent()`-method to save the intent receiver from being killed.

A unique feature of intent receivers is that the application they belong to does not have to be running when the broadcast comes in. If necessary, the Android system will start the application when an intent receiver is trigged.

An intent receiver has to specify in the applications deployment descriptor via an intent filter on what broadcast actions it should be triggered. Typical standard broadcast actions are `BOOT_COMPLETED_ACTION`, `PACKAGE_ADDED_ACTION`, `PACKAGE_REMOVED_ACTION`, `SCREEN_ON_ACTION`, `SCREEN_OFF_ACTION`, `TIMEZONE_CHANGED_ACTION`, etc. In its most current version (Build m5-rc15f), Android provides more than 40 broadcast actions.

[9] http://code.google.com/android/reference/android/content/IntentReceiver.html#onReceiveIntent(android.content.Context,\%20android.content.Intent), last viewed 2008-05-08

A complete list of all system broadcast (and activity) actions is provided in the documentation of the `Intent` class[10].

Such a powerful mechanism as broadcast actions is of course subject to security restrictions. Some broadcast actions may impose permissions in order to be received by an application. The permissions are also expressed as constants that have to be included in the applications deployment descriptor. The requested permissions are granted/denied at application install time. Either based on checks with trusted authorities (code signing and certificates) or through user interaction. More information about the Android security features can be found in the "Security and Permissions"[11] section of the online documentation.

Service Services are long lived, have no user interface and execute in the background. A typical example of a service is a media player application. Although the player UI is no more in the focus of the user, it may still be desired to keep playing the music in the background. It is important to note that services, like other application objects, run in the applications main thread. That is why they should spawn a new thread when doing any CPU intensive tasks like I/O or network operations.

Content Provider Content providers are another unique concept of the Android platform. They represent a kind of uniform wrapper over application data and are the only way for applications to share data accross packages. No matter how the data is stored internally, like e.g. as a SQLite database, in a file, or over the network, it can be exposed to other applications via a uniform content provider. Most of Android's built-in applications like e.g. contacts, SMS/MMS, calendar, etc. publish their data over content providers.

A content provider offers all CRUD (Create, Read, Update, Delete) operations on its underlying data through its `insert()`, `query()`, `update()` and `delete()` methods. They are accessed very similar to database records as they also return a cursor in their `query()`-method.

Like activities, intent receivers and services, content providers have to be registered in the applications deployment descriptor (see 6.5.1). This is done by mapping the content providers class file to a unique *authority*. For third party applications, an authority usually is a full-qualified class name to ensure uniqueness.

Content providers can be accessed via the `ContentResolver`[12] class. This class is provided by the Android runtime systems and resolves con-

[10] http://code.google.com/android/reference/android/content/Intent.html, last viewed 2008-05-07

[11] http://code.google.com/android/devel/security.html, last viewed 2008-05-07

[12] http://code.google.com/android/reference/android/content/ContentResolver.html, last viewed 2008-05-15

tent providers by means of a unique *content URI* identifier. The authority together with a standard prefix is the base part of an URI and used to match an authority to a concrete provider class. It is roughly equivalent to a database URL known from relational databases. The subsequent parts of a content URI address the sets of data that can be accessed, as well as indivdual records. This can roughly be compared to tables in a relational database respectively individual rows.

In order to clarify things a bit, figure 6.6 shows a sample content URI taken from the Android online documentation[13]:

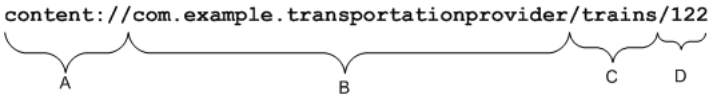


Figure 6.6: Exemplary Android content URI

A) The standard prefix, which is always the same.

B) The authority of the content provider. Under this exact name the content provider is registered within the Android system. If the content provider class name of the provider was `TransportationProvider`, the registration entry in the deployment descriptor could look like:

```
1 <provider class="TransportationProvider" authorities="com.example.
      transportationprovider" />
```

C) The path to a particular set of data. This is roughly equivalent to a table in a relational database.

D) A specific record being requested. This maps to a unique id-field in the providers data set. If this last part is omitted, all records of a particular data set are returned.

What is important to keep in mind is that the Android system is not aware of the structures beyond the authority. From the reference figure 6.6, only parts A and B are used for mapping the provider instance in the deployment descriptor. This consequently means that application developers need to make their full content URIs somewhere publicly available. As a convention suggested by the Android online documentantion, this should be done in a constant field of type `Uri`[14] and the name `CONTENT_URI`, which

[13] http://code.google.com/android/devel/data/contentproviders.html, last viewed 2008-05-07

[14] http://code.google.com/android/reference/android/net/Uri.html, last viewed 2008-05-15

ensures that the URIs can easily be found in the Java documentation of the application. As will be shown in section 6.5.4 about the realization of the prototype, this convention is a key part for finding all content providers of third-party applications at runtime.

The AndroidManifest.xml File

The AndroidManifest.xml describes global values for a package, like amongst others the application components (activities, intent receivers, services, content providers) their corresponding intent filters as well as security restrictions and requested permissions. Although the Android online documentation[15] does not use the notion of a deployment descriptor, the AndroidManifest.xml is conceptually similar to the *web.xml* deplyoment descriptor found in Java Enterprise Edition web applications. The file is always located in the root folder of an application and the name AndroidManifest.xml must not be changed.

The complete AndroidManifest.xml file of the prototype application is shown in listing 6.7 on page 121.

Application Life-Cycle

Another distinct feature of the Android system is its – compared to other operating systems – unorthodox application life-cycle management. Like on other Linux systems, every Android application normally runs in its own process. The unusual behavior of the Android system is that the applications process's lifetime is *not* directly controlled by the process itself. Instead the Android system pursues a pre-emptive strategy which can result in the killing of particular processes if the system is low on memory. The system uses a mixture of the parts of an application it knows are running, how important the application is to the user and the overall system memory to determine the process's lifetime.

The application objects that influence the process life-cycle are `Activity`, `Intent Receiver` and `Service`. The Android systems places these application objects in an "importance hierarchy" to determine which ones should be killed when the system is low on memory. According to the online documentation[16], the order of importance is as follows (beginning with the most important type of process to the least important one):

1. **Foreground Processes**
   A foreground process is the most important type of process in the Android system. This is a process that is immediatelly important to the user. An application process is considered a foreground process if one of the following conditions hold:

[15] http://code.google.com/android/devel/bblocks-manifest.html, last viewed 2008-05-08
[16] http://code.google.com/android/intro/lifecycle.html, last viewed 2008-05-08

- It is running an `Activity` on top of the screen.
- It has an `IntentReceiver` running in its `onReceiveIntent()`-method
- It has a `Service` currently executing code in one of its callbacks (`onCreate()`, `onStart()` or `onDestroy`).

The system will only kill such a process under *extrem low* memory conditions (memory paging).

2. **Visible Processes**
   A visible process is one that currently has an activity running visible to the user but not running in the foreground. This can e.g. be if a foreground activity has a dialog appearence and the previous foreground activity is still visible to the user. Such a process (the one still being visible to the user but not being in the foreground) is considered extremely important by the system and will only be killed if it is required to keep all foreground processes running.

3. **Service Processes**
   A service process is one holding a service that is currently executing some code. This could e.g. be the background music played in the service of an mp3-player application. These processes usually perform things the user cares about, so they will only be killed if there is not enough memory to keep alive all foreground and visible processes.

4. **Background Processes**
   A background process is one that holds an activity that is currently not visible to the user. Furthermore it is also not currently executing code in a service or the `onReceiveIntent()`-method of an intent receiver. The system will kill such a process at any time it needs to reclaim memory. The choice of which process to kill exactly is made based on a Least Recently Used (LRU) algorithm. For the application developer this means that the activity life-cycle has to be implemented correctly to avoid data loss or corruption.

5. **Empty Processes**
   An empty process doesn't hold any active application components. Empty processes could also be killed immediately, but they are kept by the Android system as a cache to speed up startup time if they are still in the memory the next time they are called. Quite clearly the system will instantly kill such a process if it requires memory.

As a direct consequence to the Android application life-cycle management, the application developer has to be aware of how to react to a situation when the system kills an application process and design the affected classes accordingly.

Another interesting fact is that Android runs *all* application objects (activities, intent receivers and services) in the process's main thread. This means that it is the application developer's responsibility to provide the necessary threading to keep the application responsive and avoide a mutual blocking of individual components.

6.5.2 Prototype Architecture

The architecture of the Android native client is divided into two main parts: The *service consumer* and the *content provider crawler*.

The service consumer fulfills the same functions as the mobile AJAX client. The content provider crawler automatically scans all installed applications for available content providers and pushes the selected contents to the Mindbreeze Indexing Service. Since the Mindbreeze Indexing Service does not yet support the pushing of data, the push index for testing the prototype is simulated by an upload servlet.

Service Consumer

The main purpose of the the service consumer part of the Android prototype is to act as a mobile enterprise search client to the Mindbreeze Query Service. Like the web client, it supports pagination for breaking bulky results into several pages. A screen shot of the user interface is shown section 6.5.3.

Equally to the iPhone client it uses the MES Query Service which means that it handles the same data exchange formats (see section 6.3). Since the Android platform does not out of the box support a concept similar to that of the AJAX XMLHttpRequest object, a non-blocking search request cycle has been modeled.

Figure 6.7 shows the class diagram of the classes involved from issuing the request to displaying the results to the user in a list. For the sake of clarity, only the key fields and methods of the involved classes are shown.

In order of execution, the following steps are performed when issuing a request:

1. The user types a search phrase into a text box in the `SearchClient` class. `SearchClient` is the main activity of the prototype. It is lauchend when the application is started and runs in the main thread of the application process. It is a subclass of `ListActivity`[17], which means that it is tailor-made for activities exposing lists.

2. As soon as the user issues the request, the input data is stored in a `SearchRequest` object. This object can be serialized either as XML string for being included as payload in an HTTP-POST request, or

[17] http://code.google.com/android/reference/android/app/ListActivity.html last viewed 2008-05-12

as query string parameters for use in an HTTP-GET request. In the case of an XML serialization, a simple self written `XMLElement` class is used for constructing the XML document, to avoid the need for a potentially heavy-weight XML library.

3. The search request is then enqueued in the `AsyncRequestManager`. This class provides functionalities similar to that of the AJAX XMLHttpRequest object. It leverages the new concurrency features introduced in Java 1.5, which are also available in Android and found in the package `java.util.concurrent`. The `Executor` interface and the `ExecutorCompletionService` class build a thread pool. The pool only spawns one thread. If multiple requests are being made within a short period of time, they are queued and are processed as soon as the executor thread gets available. The class for actually conducting the request, `HTTPRequestWorker`, implements the `Callable` interface and gets calculated from the executor thread in its `call`-method. As soon as the server has finished the request the returned HTTP status code and the input stream (response body) are stored in an `HashMap`, where the `AsyncRequestManager` can "pick" them up. The `AsyncRequestManager` class itself runs in a separate thread and keeps waiting for finshed requests in its `run()`-method. Due to the non-blocking nature of the `take()`-method from the `ExecutorCompletionService` class, the object waits (goes to sleep) until there is data available and hence avoids busy waiting.

4. Every call to a method that updates the user interface must be made from the main UI thread (for more details see the Android online documentation[18]). Since the `AsyncRequestManager` spawns its own thread, it is not possible to directly call the `SearchClient` class to update the user interface. As a solution, Android offers the `Handler`[19] class which gets started on the UI thread (`SearchClient`) and allows for passing messages to its message queue. The passed messages have to be of type `Message`[20], which is a wrapper for arbitrary data stored in a `HashMap`.

5. The processing of the message data is done in the handler's `handleMessage()`-method. The returned response payload string is read from the message and given to an instance of the `SearchResponseParser`.

[18] http://code.google.com/android/reference/android/view/View.html, last viewed 2008-05-12

[19] http://code.google.com/android/reference/android/os/Handler.html, last viewed 2008-05-12

[20] http://code.google.com/android/reference/android/os/Message.html, last viewed 2008-05-15

6. The `SearchResponseParser` is a subtype of `DefaultHandler`[21] and does the parsing of the response XML document by means of the SAX parser provided by the `org.xml.sax` and `javax.xml.parsers`. It wraps the parsed data in the `Hit` and `HitContextMenu` objects and adds them to the respective methods provided by the `ResultListAdapter` class.

7. Finally the custom adapter `ResultListAdapter` which extends `BaseAdapter`[22] and is attached to the `SearchClient` class is used to update the search client user interface.

[21] http://code.google.com/android/reference/org/xml/sax/helpers/DefaultHandler.html, last viewed 2008-05-15

[22] http://code.google.com/android/reference/android/widget/BaseAdapter.html, last viewed 2008-05-12

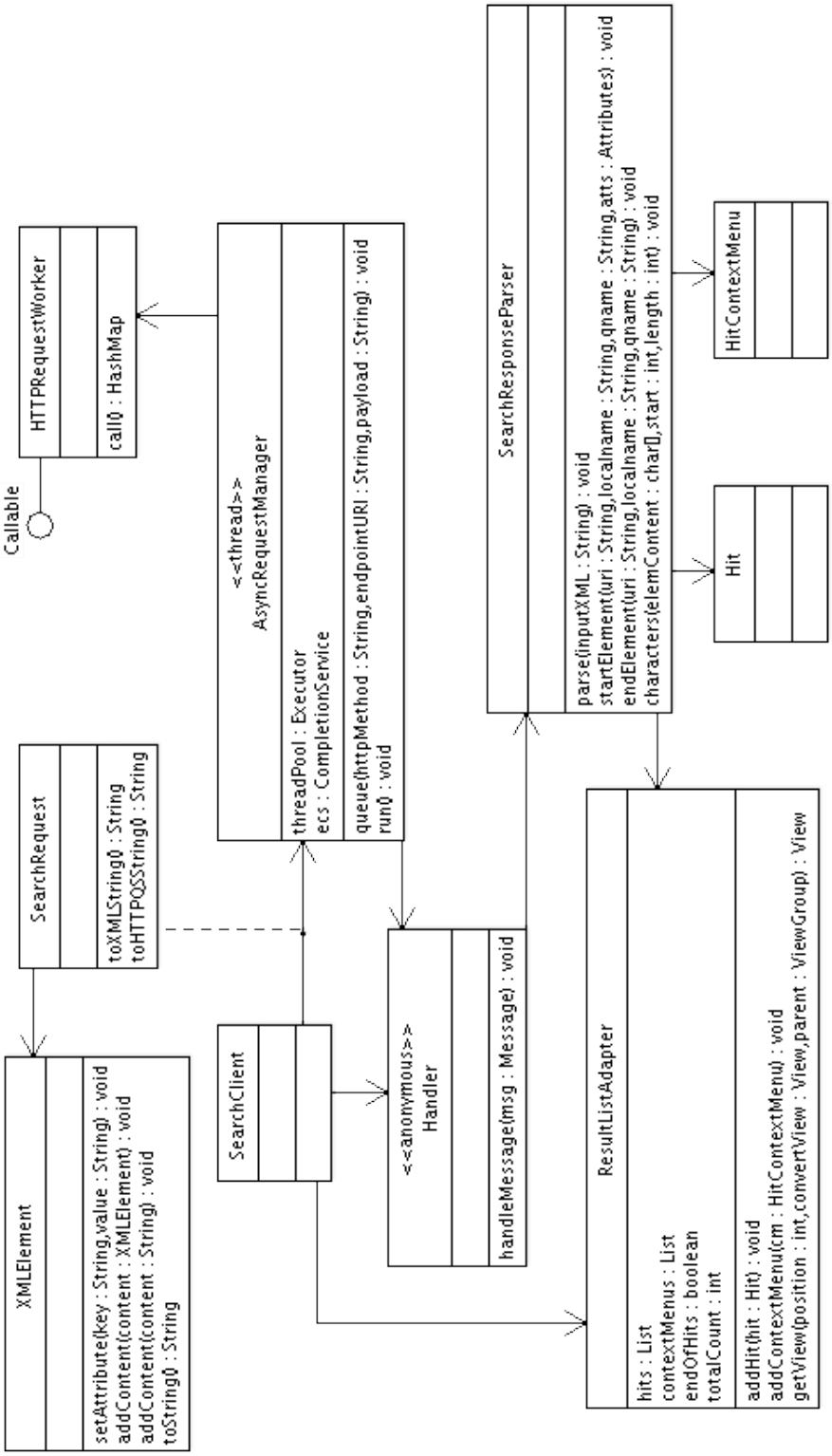


Figure 6.7: Android client class diagram of service consumer part

Content Provider Crawler

The content provider crawler is designed to perform the following tasks:

- Discover all content providers of all installed applications (built-in as well as third-party applications).
- Query their data and push it to the Mindbreeze Indexing Service.
- Keep track of changes and modifications to the providers and sychnronize the changes with the Indexing Service.

A more detailed view of the functioning of the content provider crawler is best provided by means of two of the most important use cases. The installation of the application and the first login (see figure 6.8), as well as the "normal" operation once the application is installed (see figure 6.9).

Installation and First Login The most important task the crawler has to perform once the user has authenticated herself, is to scan the device for any content provider exposed by the currently installed applications. As figure 6.8 shows this is done by starting the `ProviderScannerService`. The service first makes a full scan of all applications installed on the device and stores the most important information about the found applications, their content providers and the providers content URIs via the `ContentProviderAdapter` in a local SQLite 3 database. More information about the content provider discovery and how it is done programmatically is provided in section 6.5.4.

The resulting content providers are then registered in the long lived `ChangeListenerService` which constantly monitors them with regard to updates and modifications. The `ChangeListenerService` is the longest lived of all services used in the content provider crawler application. It is started immediately after the first successful login and will under normal circumstances only be stopped when the search client application gets uninstalled. This is a key difference to all the other services of the application, as they only exist as long as they actively perform a task.

As soon as all installed applications have been searched for content providers, the list of discovered providers is displayed to the user, who then has the possibility to selectively choose the providers she is willing to be queried by the crawler.

Alongside the initialization of the `ProviderScannerService` and the `ChangeListenerService`, the `AlarmManagerReceiver` intent receiver is created. This receiver is registered for repeating alarms sent by the Android system via the `AlarmManager`[23]. The `AlarmManager` is a scheduler that al-

[23] http://code.google.com/android/reference/android/app/AlarmManager.html, last viewed 2008-05-14

lows an application to be run at some point in the future. The target application does not have to be running to receive the intent, because it will be started by the Android system if it is not currently alive. The system may optionally even wake up the device if it is sleeping while the alarm goes off.

In case of the Mindbreeze Android client, the `AlarmManager` is scheduled to start the `ProviderQueryService` every three hours. Like the `ProviderScannerService`, the `ProviderQueryService` is short lived and executes in a separate thread to avoid blocking of the main thread. Its task is to check the database for newly added or changed providers. If there are some, it always queries the *entire provider*, even in case of a change to an already queried provider[24], and pushes all data to the server index. Finally the timestamp of the query and similar meta information is stored in the local database.

[24] Through the Android `ContentObserver` class it is only possible to detect that a content provider has changed at all, not what has been changed.

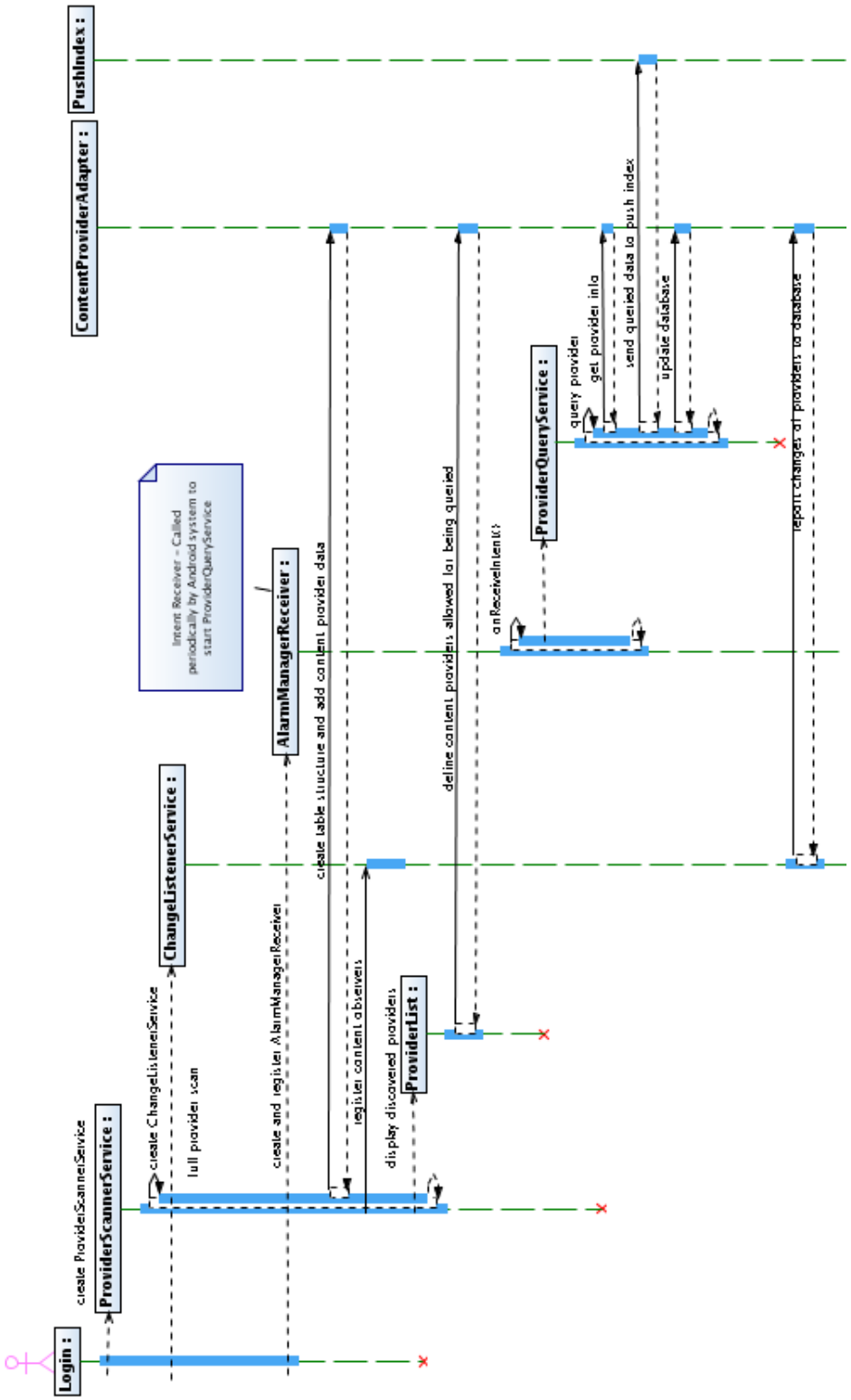


Figure 6.8: Sequence diagram of Android content provider crawler at first login

Normal Operation The sequence diagram of the client in normal operation shown in figure 6.9 is essentially very similar to that depicted in figure 6.8. A few things need to be considered compared to the installation and first login use case.

Firstly, the client application possesses the `BootCompletedReceiver` intent receiver which gets called by Android after the system has finished booting. The purpose of this class is to start the long lived `ChangeListenerService` that monitors all discovered content providers for changes as well as the `AlarmManagerReceiver` that will regularly start the `ProviderQueryService`. These tasks have been performed by the login routine in the first login use case. Launching these functionalities at device boot time guarantees that they are always running.

In contrast to the launching of the `ChangeListenerService` in the first login use case, this time all content providers already stored in the database are registered at start time in the services' `onCreate()`-method. This is necessary because by contrast to intent receivers, content observers are normal Java event listeners and hence have to exist at runtime in order to receive events.

Another additional component is the `PackageActionReceiver` intent receiver. It is used to keep track of newly added or removed packages. Whenever a new package gets added to the system, Android calls this class which starts the `ProviderScannerService` that searches the newly added package for any content provider. The procedure of the `ProviderScannerService` is equivalent to that performed at initialization of the application, with the only difference being that only one package is searched instead of all.

In case the system sends a package removed broadcast action, the `PackageActionReceiver` launches the `DeletionService` which performs a diametrically opposed task compared to the `ProviderScannerService`. It removes the content observers of the uninstalled package from the `ChangeListenerService` observer list, informs the Indexing Service about the deletion and finally erases all artifacts of the application from the local database.

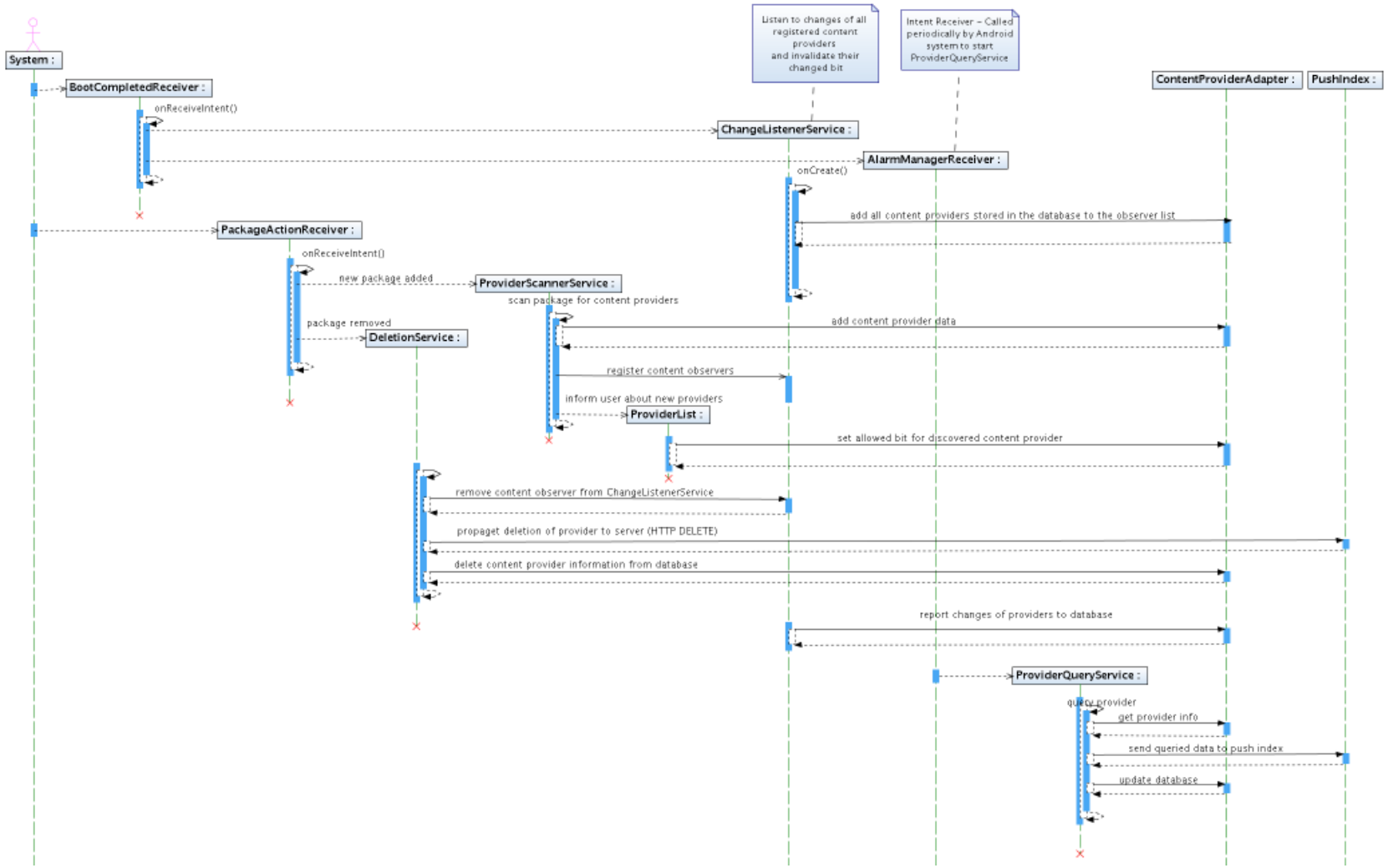
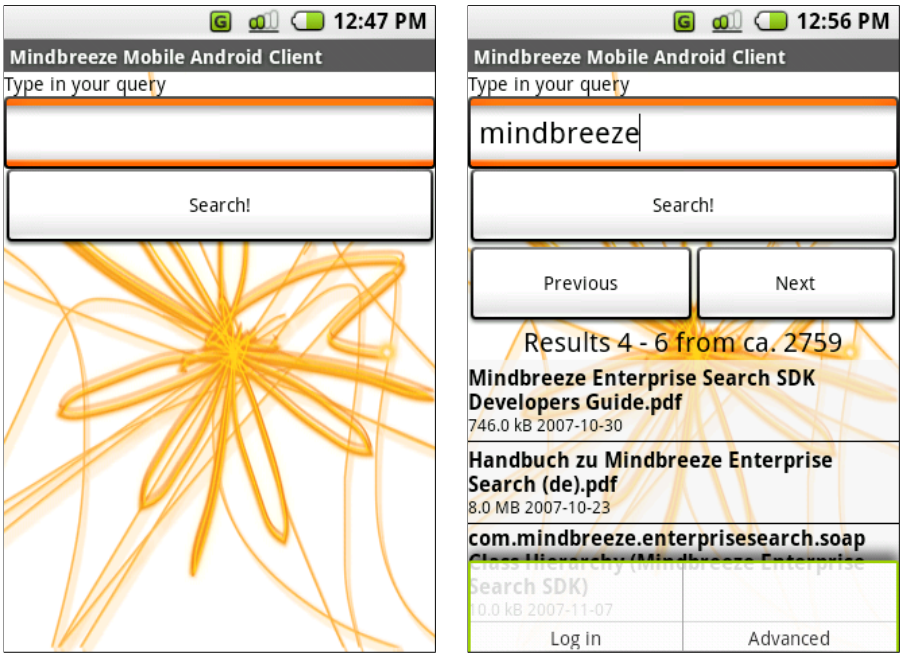


Figure 6.9: Sequence diagram of Android content provider crawler in normal operation



(a) (b)

Figure 6.10: Android client directly after launching (a) and after the search request has been processed by the server (b)

6.5.3 Graphical User Interface (GUI)

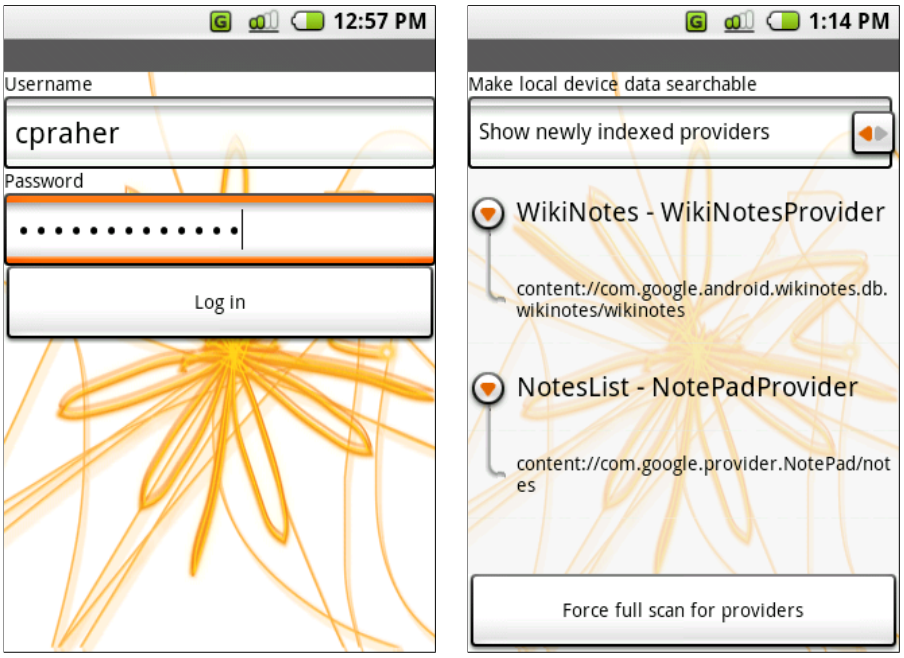
In contrast to the iPhone client, the focus of the Android native client has not been on the Graphical User Interface (GUI). It is a proof of concept to investigate what possibilities a native mobile platform offers in respect to leveraging the device APIs in an enterprise search scenario.

Figure 6.10 (a) shows the application directly after being launched. The most apparent difference of the starting screen compared to that of the iPhone client (figure 6.3) is the presence of a search button. This stems from the fact that before the availability of the first officially released Android handsets it is hard to judge exactly what device form factors will be available and how they are best treated in means of data input.

Figure 6.10 (b) depicts a screenshot of the client after the search request has been processed and the results have been presented to the user.

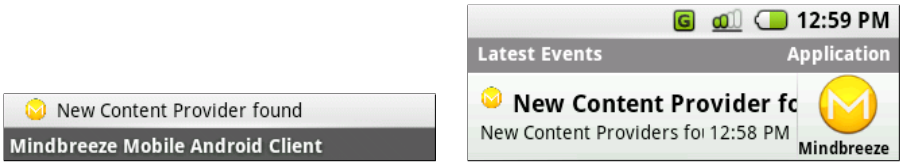
The option menu buttons titled "Log in" and "Advanced", shown at the bottom of figure 6.10 (b), appear if the user presses the menu button of the phone. Clicking "Log in" takes the user to the login screen depicted in figure 6.11 (a), the "Advanced" button opens the provider list activity shown in figure 6.11 (b).

Whenever an action occured that is relevant to the user, like e.g. a new



(a) (b)

Figure 6.11: Android client login screen (a) and overview list of discovered content providers (b)



(a) (b)

Figure 6.12: Android client status bar notification (a) and pull down menu for navigating to the action intent of the notification (b)

content provider has been found, or the querying of the content providers has been finished, or an application with a registered content provider has been uninstalled, the notifaction shown in figure 6.12 (a) is displayed in the status bar. If the user pulls the notification down, or presses a special purpose button in case of a non touch screen device, the notification pull down depicted in figure 6.12 (b) is shown. This menu contains an additional description about the occurred event, as well as a timestamp for knowing when it happend. By pressing the icon on the right or the description text on the left, the user is taken to the activity associated with the notification.

6.5.4 Realization

After the overview of the content provider crawler and the presentation of its GUI, this section aims at providing an insight into the inner workings of some of the parts of the application. Notably the AndroidManifest.xml deployment descriptor, the underlying database schema as well as the content provider discovery mechanism will be presented in more detail.

AndroidManifest.xml

As mentioned in section 6.5.1, every Android application has an AndroidManifest.xml deployment descriptor that describes the most important values, classes and permission of an application. Listing 6.7 shows the AndroidManifest.xml of the Mindbreeze Android crawler client application.

```
1 <?xml version="1.0" encoding="utf-8"?>
2 <manifest xmlns:android="http://schemas.android.com/apk/res/android"
      package="com.mindbreeze.mobile.android">
3
4     <!-- The NEEDED PERMISSIONS -->
5     <uses-permission android:name="android.permission.
          RECEIVE_BOOT_COMPLETED" />
6     <uses-permission android:name="android.permission.READ_CONTACTS" />
7
8     <application android:icon="@drawable/mindbreeze_icon_64x64_transp">
9
10      <!-- The ACTIVITIES -->
11      <activity android:name=".SearchClient" android:label="@string/
            app_name">
12        <intent-filter>
13          <action android:name="android.intent.action.MAIN" />
14          <category android:name="android.intent.category.LAUNCHER" />
15        </intent-filter>
16      </activity>
17      <activity android:name=".login.Login" />
18      <activity android:name=".advanced.ProviderList" />
19
20      <!-- The SERVICES -->
21      <service android:name=".advanced.ChangeListenerService" />
22      <service android:name=".advanced.ProviderScannerService" />
23      <service android:name=".advanced.ProviderQueryService" />
24      <service android:name=".advanced.DeletionService" />
25
26      <!-- The INTENT RECEIVERS -->
27      <receiver android:name=".advanced.BootCompletedReceiver">
28        <intent-filter>
29          <action android:name="android.intent.action.BOOT_COMPLETED"/>
30        </intent-filter>
31      </receiver>
32      <receiver android:name=".advanced.PackageActionReceiver">
33        <intent-filter>
```

```
34          <action android:name="android.intent.action.PACKAGE_ADDED"/>
35          <action android:name="android.intent.action.PACKAGE_REMOVED"/>
36          <data android:scheme="package" />
37        </intent-filter>
38      </receiver>
39      <receiver android:name=".advanced.AlarmManagerReceiver" />
40
41    </application>
42 </manifest>
```

Listing 6.7: Android prototype – AndroidManifest.xml File

Directly after the XML declaration and the `<manifest>`-root-tag that defines the applications Java package name, is the definition of the required permissions, which are the permission to receive system boot broadcast actions as well as the permit to read data from the built-in contacts provider.

The next tag is the `<application>`-tag which summarizes all classes of the application that directly interact with the Android runtime system, including activities, services and intent receivers. The class name of each component is denoted by the `android:name` attribute. It can either be provided directly as a full qualified Java class name or beginning with a dot, which results in a full qualified class name as the concatenation of the `package`-attribute of the `<manifest>`-tag and the respective class name fraction specified by the components `android:name`-attribute.

As figure 6.7 shows, the search client application consists of three visual activities that are specified via `<activity>`-tags. As appareant through the `<intent-filter>`-child-tags, the `SearchClient` activity is the applications entry point and hence launched when the application is started.

Following the activity tags are the `<service>`-tags which list all the services used by the application. It is import not to forget to list a service in the deployment descriptor, because otherwise it can not be started.

Finally, the intent receivers of the application that receive the desired system broacast actions are listed in the `<receive>`-tags. The broadcast actions that trigger the launch of the intent receiver is declared in the `<intent-filter>`-tags. The `AlarmManagerReceiver` does not have an `<intent-filter>`-child-tag, because it is directly configured as intent of the `android.app.AlarmManager` class at the initial login of the application, respectively each time the device has finished booting.

Database Schema

The database schema of the Android search client is held intentionally very simple. Figure 6.13 shows that is used for storing the most important data about applications, their associated content providers and the concrete content URIs, which allow for querying the providers. The mechanism of dis-

covering the data described in this data model is explained in more detail in the next section (section 6.5.4). Content data gathered from querying the content providers is *not* stored in the local database. This information is only pushed to the Mindbreeze Indexing Service.

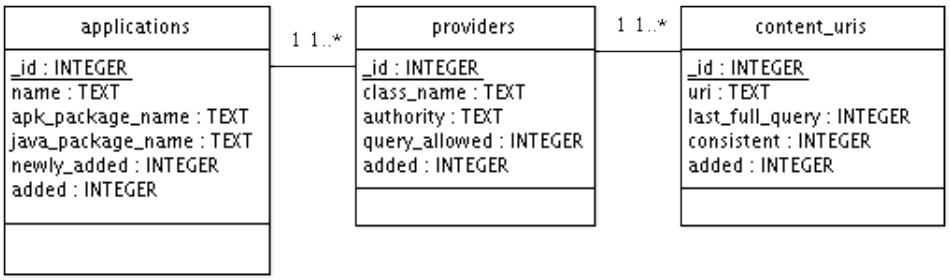


Figure 6.13: Database schema of the Android search client

An Android application does not necessarily need to have a content provider. E.g. the Mindbreeze search client does not expose any data to other applications and hence does not possess a content provider itself. Still the cardinality between applications and providers in the schema is one to at least one (1 1..*). This is due to the fact that only applications exposing a content provider are stored in the local database. As a consequence all applications listed in the database do at least have one content provider.

The rough relationship between the Android content provider and the relational database model is shown in table 6.1.

Android Model	Relational Model
Application	Application
Content Provider	Database
Content URI	Table

Table 6.1: Comparison between Android content provider and relational model

Content Provider Discovery Mechanism

The discovery of the content providers and moreover the content URIs of applications installed on the device is the key part of the content provider crawler. Responsible for this task is the `ProviderScanner` class, respectively the `ProviderScannerService`, which calls the methods of the `ProviderScanner`

in its service routines.

Content providers are defined in an applications AndroidManifest.xml file via the `provider` tag (see the paragraph about content providers in section 6.5.1). The two mandatory attributes are `android:name` which holds the name of the class that implements the provider as well as `android:authorities` which serves as a unique identifier of the provider within the Android system. Every provider implementation must extend the abstract class `ContentProvider`[25] which guarantees a uniform interface of every provider and allows the system to programmatically query them via the `ContentResolver` class.

The key class to retrieving information from packages installed on the device is the `PackageManager`[26]. It offers methods for getting information about all applications currently installed on the device as well as methods for retrieving the data of one particular package.

Almost all fields shown in the database model in figure 6.13 can be extracted from the methods offered by the `PackageManager` class. This includes the application name, the APK and Java package name, as well as the content providers class name and its authority.

Besides the metadata, only one essential field shown in the data model still is missing: the *content URI*. The content URIs needed to access content providers are not known to the Android system and hence can not be gathered from the `PackageManager`. From the sample content URI shown in figure 6.14 only the system part is known to the system. This is a direct consequence of the fact that only the authority part is used to register a certain provider. Any queryable information exposed by the content provider is *not* known to the Android system and the exclusive responsibility of the application.

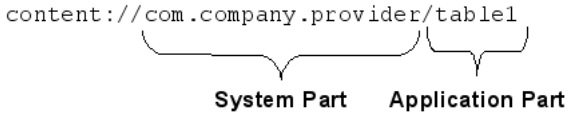


Figure 6.14: Android content URI divided into system and application part

Typically an application developer who wants to query a content provider table takes the full qualified content URI (like e.g. the one showed in figure 6.14) and calls the `query` method of the `ContentResolver` class. The An-

[25] http://code.google.com/android/reference/android/content/ContentProvider.html, last viewed 2008-05-15

[26] http://code.google.com/android/reference/android/content/pm/PackageManager.html, last viewed 2008-05-15

droid system then resolves the content provider implementation referenced by the system part of the given URI and calls its `query` method with the full content URI as parameter. Within the `query` method of the particular content provider, it is the application developer's responsibility to choose the desired table (and optionally a specific record) denoted by the application part of the content URI and return the desired cursor.

As a consequence, the full content URI to a particular table has to be known before querying the table. Since the Android system does not know about the various tables inside a specific content provider, there does also not exist a method for getting all table names at runtime like it is for example possible with the SQL `show tables` command.

Normally this is not a problem, because the creator of a content provider has to document all content URIs. As mentioned in section 6.5.1, the convention proposed by the Android online documentation is to make them public in a static field of type `Uri` named `CONTENT_URI`.

Given that the content provider crawler needs the information about content URIs at runtime in order to be able to generically query them, it has to reflectively search all[27] classes contained in a package for `CONTENT_URI` fields and read their values.

Reflectively find the CONTENT_URI fields Although Android features the standard Java reflection packages (`java.lang.reflect`) it proved to be impossible, despite various different experiments, to (recursively) read all classes contained in an applications Java package. As an alternative an indirection via the Android classes.dex file, which is contained in every Android APK package, has been taken. The dex file is the executable used by the Android Dalvik JVM. By contrast to normal Java bytecode files (.class-files) where every Java source file is represented by one binary file, the entire bytecode of an Android application is contained in one single dex file. Amongst other information the dex file also includes a class list which contains all classes referenced or contained in the dex file. This class list is used by the `ProviderScanner` to retrieve all class names of a given application package.

As at the time of writing of the search client application the Dalvik JVM and the dex file format have not yet been open sourced, a description of the dex file format provided by the community[28] served as the basis for extracting the class names. The following list provides a schematic overview of the steps involved in discovering the content URIs of an application:

1. The only thing known at the beginning is an applications Java pack-

[27] The Android documentation makes no recommendation about where to put the `CONTENT_URI` fields. The only thing granted is that they have to be somewhere in the package of the application that exposes the provider.

[28] http://www.retrodev.com/android/dexformat.html, last viewed 2008-05-15

age name. This is either retrieved from the `getInstalledPackages()` method of the `PackageManager` class in case of a full scan, or via the package added broadcast action for applications that get installed after the crawler is first used.

2. With the help of the `getPackageInfo()` method of the `PackageManager` class, all of an application's content providers and their authorities can easily be retrieved.

3. The next step is to load the context of the third party application with the `createPackageContext()` method of the `Context`[29] class.
Once the context is loaded, its Android package file path can be elicited with the `getPackagePath()` method of the context instance.

4. Since an Android package is a zip-file, it can be unzipped with the Java zip utilities provided by the `java.util.zip` package, which is part of the Android platform.

5. The Android dex (classes.dex) file, which is always included on the top level of an Android package (.apk) can then be stored in a temporary file in the own context. Writing the zip input stream to a temporary local file is necessary for being able to open it as Java `RandomAccessFile`.

6. Following, the class file names can be read from the dex file according to the description provided by the unofficial Android dex file documentation. The fields needed to find the class names are given in tables 6.2, 6.3 and 6.4. Attention needs to be paid to the fact that Android stores bytes in *little endian* order, which is opposed to the standard Java *big endian* byte order format. The method for converting a byte array to an integer number is shown in listing 6.8.

7. Once all class names are collected, they can be loaded and reflectively scanned for `CONTENT_URI` fields by means of "normal" Java reflection. To be discovered as `CONTENT_URI` field, all attributes described by the Android online documentation have to hold, which are:

   - The field has a `static` modifier
   - Its name equals `CONTENT_URI`
   - Its field type is `android.net.Uri`

8. In order for a content URI to be matched to a particular content provider, its value has to be read and the authority-part of the content URI has to be matched to one of the providers of the application.

[29] http://code.google.com/android/reference/android/content/Context.html, last viewed 2008-05-15

9. Finally, the found `CONTENT_URI` can be added at the right place in the `content_uris` table.

Offset	Size	Description
0x30	4	Number of strings in the string table
0x34	4	Absolute offset of the string table
0x3C	4	Number of classes in the class list
0x40	4	Absolute offset of the class list

Table 6.2: Android dex file – File header

Offset	Size	Description
0x0	4	Absolute offset of the string data
0x4	4	Length of the string (not including the null-terminator)

Table 6.3: Android dex file – String table

Offset	Size	Description
0x0	4	Index of the name of the class in the string table

Table 6.4: Android dex file – Class list

```
1  /**
2   * Helper function for converting a byte-array of size 4
3   * into a 32-bit integer.
4   * The bytes are assumed to be stored LITTLE-ENDIAN
5   * (Android Dalvik Style).
6   * Most significant byte comes last - Least significant first.
7   */
8  private int bytesToInt(byte[] bytes) {
9    //The 0xFF mask normalizes signed bytes (-127 to 128) to
10   //an unsigned range from 0 - 255 and converts them to short (integers)
11   int number = ((bytes[3] & 0xFF) << 24)
12     | ((bytes[2] & 0xFF) << 16)
13     | ((bytes[1] & 0xFF) << 8)
14     | (bytes[0] & 0xFF);
15
16   return number;
17 }
```

Listing 6.8: Android prototype – `bytesToInt()`

6.5.5 Deployment

As at the time of writing of this thesis no handsets built on the Android platform have yet been available, the various ways of deploying an application on an Android phone in a real life scenario are not yet known.

The following steps are needed to manually deploy an application on the emulator:

1. Compile the Java source files with a normal Java compiler
2. Convert the Java class files (.class) to a Dalvik executable (.dex) with the `dx` tool
3. Create the necessary folder structures and create an Android package (.apk) with the Android Asset Packaging Tool `aapt`
4. Install the package on the device with the Android Debug Bridge `adb` tool. The command for installing a package on the device is `adb install <apk_package_name>`.

Further information is provided in the tools section of the Android online documentation[30].

The above steps are only needed in case of manually deploying the application on the emulator. The Android SDK also offers an Eclipse IDE plugin which reduces the overhead of packaging and installation of the application to simply pressing the IDE's "run' button.

A deployed package is uninstalled simply by removing it from the emulator. This is done by accessing the device's shell with the `adb shell` command[31]. All installed application packages are located in `/data/app/`. The application that should be deleted can then be simply uninstalled by removing it's Android pacakge with the standard Linux remove command `rm app_to_remove.apk`.

[30] http://code.google.com/android/intro/tools.html, last viewed 2008-05-15

[31] The emulator has to be running for this command to succeed

Chapter 7

Summary & Conclusion

This paper covers a wide range of topics in the realm of mobile computing and can basically be divided into two major parts. A theoretical part providing an introduction to mobile computing, smartphone operating systems and development platforms, as well as mobile web applications and mobile Service Oriented Architecture (SOA). And a practical part that shows two distinct prototype implementations of a mobile enterprise search client.

The first part aims at providing an overview and introduction into the rapidly changing field of today's mobile computing. It tries to answer question such as "What are the most dominant mobile operating system and development platforms?", "Is there a platform or technology that targets many or maybe even *all* mobile devices?", "What is the status quo of today's mobile application and web application development?", "How could a mobile SOA application be realized?", "What possibilities of accessing the device context and the locally stored data do the individual technologies offer?", etc.

One of the key findings of this part of the thesis is that the mobile market is much less consolidated than that of desktop computers. The desktop operating system market is clearly defined with only a few serious competitors and an almost monopolistic dominance of the Microsoft Windows family of operating systems. By contrast, the mobile operating system market is currently subject to rapid changes and many competitors are fighting fiercely for market shares in this promising area of computing. As of today, it is extremly hard if not impossible to predict which platform will eventually prevail. Although table 3.1 on page 14 about the market shares of smartphone operating systems suggests a clear market leadership of Symbian OS, it has to be questioned if this is as clear as it seems at first sight. The smartphone operating system market is regionally very diverse. According to Symbian Ltd.'s smartphone market analysis of quarter 3 of 2007 [72], Symbian OS had a market share of more than 80 % in Europe and Middle East, while holding less than 10 % of the North American market. The presented

platforms are often subject to internal fragmentation. In case of Symbian OS, there exist three different GUI platforms. In case of mobile Linux one can not talk about a single platform at all. It is rather split up in many different derivatives, of which most have different application development models. The mobile market is still in its infancy, which means that new platforms may have big impacts on the overall market situation and market shares are subject to much faster changes than in more traditional markets like that of desktop computers. A good example for this is Apple's iPhone that already gained a worldwide market share of 7 % in the last quarter of 2007 [11], although it had only be released in July of the same year.

Chapter 3 shows that Java ME is available on a large number of systems and as a middle layer technology it possesses high potential for an operating system independent application development platform. Yet, especially the Java ME platform is highly fragmented and offers significantly different possibilities concerning mobile application development depending on the particular device and the Java ME version.

A similar situation as with mobile operating systems can be seen in the area of mobile browsers. While the desktop market again shows clear market proportions, the mobile browser market is characterized by a high fragmentation. As outlined in chapter 4, the dominant desktop browsers are not found with an equal significance on mobile devices.

While the mobile market is economically characterized by a fierce competition and a big divergency resulting out of it, mobile devices rapidly mature and mobile technologies more and more converge with existing desktop technologies. This trend is apparent in the realm of mobile web applications. Still some years ago mobile web sites had to be created with a total different set of tools than their desktop counterparts. New mobile browsers nowadays (almost) support the same set of technologies as desktop browsers. This tendency can also be observed in case of native development platforms. E.g. Open Handset Alliance's Android platform offers a set of Java libraries that can rather be compared to Java Standard Edition (SE) than to the Java Micro Edition (ME). A similar trend can e.g. also be observed in case of the iPhone SDK, which supports a special version of their Cocoa application (development) environment also found on the desktop operating system Mac OS X.

Another key aspect of the fist part of the paper was to highlight the various levels of device access of the different application development technologies. Gathering information from the device and its context is crucial to many mobile applications. As presented, this is one of the major drawbacks of mobile web applications. Just like their desktop counterparts, they are strictly sandboxed and do not allow for an interaction with the hosting device. While this may change in the future as described in section 4.3.3, the only possibility for taking full advantage of the device context at present is to use a native programming language, or to some degree Java ME.

A further important finding of the paper is that simple Web Service implemenations on basis of HTTP are both possible with modern mobile web technologies (AJAX) and native as well as Java ME applications. As chapter 5 explains, the REST approach possesses several advantages over the SOAP paradigm when it comes to mobile computing. Most importantly the fewer technical requirements and the less verbose and hence faster data transmission.

The second part tries to practically demonstrate some real implementations of the technologies and trends discussed and explained in the preceding chapters. The reason behind the choice of the two platforms lies with the idea to explore the two opposite sides of today's mobile application development spectrum, which ranges from web to native applications. The clear benefits of the mobile AJAX client are its relative ease of implementation and the unbeatable easy deployment. Its weakest point and at the same time the ultimate strength of the native client is the access of the device data. With the Android core libraries and application framework the application developer has all device data at hand. Any information ranging from Personal Information Management (PIM) over mass storage card to sensing data like Global Positioning System (GPS) data can be read from the device.

7.1 Prototype Enhancements & Future Work

The security issue is the most apparent deficiency of both prototype applications. Both clients access the Mindbreeze Query Service in "Unrestricted Public Access" mode, which would be impractical in a real world scenario. Still both technologies possess the ingredients that are necessary to build a secure distributed application. In the HTTP based scenario both operate in, this is most notably SSL/TLS support. Possible approaches on how to secure the clients are briefly presented in section 5.4.2.

Another security threat not currently dealt with is the one of "loss and theft" of the device.

As work on the prototypes will continue, the security issues will of course be expanded on in the future.

7.1.1 Mobile AJAX Web Client

The appearence of the mobile web client as presented in this paper is very much tailored for the iPhone. The underlying technologies used to communicate with the Mindbreeze Query Interface on the other hand are standards based and should work with little to no modifications involved on any other AJAX enabled mobile browser. Hence adapting the application for other mobile browsers would be the next logical step.

7.1.2 Native Android Client

Since the native Android client technologically offers more possibilties than the iPhone web client, it also has more potential for further advancements. Amongst others they could be:

- The current Android search client does not have a refinements panel. The underlying `SearchRequest` object already has refinements support. Hence refinement options could be implemented by adding the necessary GUI elements. Likewise the visual appearance could further be improved by replacing the search button with whatever mechanism fits best for a particular type of device.
- Currently data is only read from the content providers. A further enhancement could be to *write* data into them. An imaginable scenario could e.g. be to insert all of a users publicly available contact information into her Android contacts content provider. This way a user would always have all contacts on her mobile phone. Furthermore it would limit the dependency from the mobile device and could facilitate migration to other Android based phones.
- Android also offers other ways of storing data aside from content providers. Most importantly they comprise local files that have been declared to be publicly read- or writeable by other applications, as well as data stored on the mass storage card (SD-card). A futurely enhanced client could also consider this data in the search.
- An application exposing a content provider is not forced to notify the system about changes to the provider, even though this is recommended by the Android online documentation. Hence a more robust version of the prototype could periodically check the `last_full_query` time-stamp to re-query providers that have not been considered for a longer time.
- Another improvement could be to let the user specify the time intervall at which the providers are queried. Ideally the user could choose between an intervall (like currently implemented) and a fixed time (e.g. every day at 03:00 a.m.).
- The Android SDK includes optional APIs for location-based services for devices that are equipped with the corresponding hardware. The data from these APIs could be used to limit or to specially sort a search response with regard to the current position of a user.
- The application could additionally implement the `SEARCH_ACTION` activity action. This would make the Mindbreeze search client the default search application on the device.

Bibliography

[1] Allamaraju, Subrahmanyam, Ronald Ashri, Chad Darby, Robert Flenner, Tracie Karsjensand, Mark Kerzner, Alex Krotov, Alex Linde, Jim MacIntosh, James McGovern, Thor Mirchandani, Bryan Plaster, Don Reamy, Dr P G Sarang and Dave Writz: *Professional Java E-Commerce*. WROX Press Ltd., 2001.

[2] Allin, Jonathan: *Wireless Java for Symbian Devices*. Symbian Press (WILEY), 2001.

[3] Apple Developer Connection: *Dynamic HTML and XML: The XMLHttpRequest Object*, JUN 2005. URL, http://developer.apple.com/internet/webcontent/xmlhttpreq.html, last viewed 2008-02-25.

[4] Apple Developer Connection: *Know What Safari Supports on iPhone*, 2007. URL, http://developer.apple.com/iphone/devcenter/designingcontent.html, last viewed 2008-02-26.

[5] Apple Inc.: *Cocoa Fundamentals Guide*. MAR 2008. URL, http://developer.apple.com/iphone/library/documentation/Cocoa/Conceptual/CocoaFundamentals/CocoaFundamentals.pdf, last viewed 2008-04-03.

[6] Apple Inc.: *iPhone OS Overview*. 2008. URL, http://developer.apple.com/iphone/gettingstarted/docs/iphoneosoverview.action, last viewed 2008-04-03.

[7] Apple Inc.: *iPhone OS Programming Guide*. MAR 2008. URL, https://developer.apple.com/iphone/library/documentation/iPhone/Conceptual/iPhoneOSProgrammingGuide/iPhoneOSProgrammingGuide.pdf, last viewed 2008-04-04.

[8] Apple Inc.: *Safari Web Content Guide for iPhone*. FEB 2008. URL, http://developer.apple.com/documentation/AppleApplications/Reference/SafariWebContent/SafariWebContent.pdf, last viewed 2008-04-04.

[9] Bersvendsen, Arve and Ian Hickson: *Opera Platform DOM Interface Specification 1.1*, APR 2005. URL, http://oxine.opera.com/documentation/dom-interface.html, last viewed 2008-05-30.

[10] Byous, Jon: *Java Technology: The Early Years*. Website, APR 2005. URL, http://java.sun.com/features/1998/05/birthday.html, last viewed 2007-12-26.

[11] Canalys: *Canalys research release 2008/021*, FEB 2008. URL, http://www.canalys.com/pr/2008/r2008021.pdf, last viewed 2008-05-28.

[12] Carlson, Brian, Uming Ko and Bill Krenik: *The Repeal of Moore's Law?* DEC 2005. URL, http://focus.ti.com/general/docs/wtbu/wtbuviewnewsletter.tsp?templateId=6123&navigationId=11952&path=templatedata/cm/general/data/wtbmiddl/newsletter/num15_smartreflex, last viewed 2008-04-24.

[13] Carney, Bruce: *Evolving to Symbian OS v9*, 2005. URL, http://developer.symbian.com/main/downloads/papers/evolving_toV9/evolving_toV9.pdf, last viewed 2008-01-31.

[14] Chinthaka, Eran: *Enable REST with Web services, Part 1: REST and Web services in WSDL 2.0*. MAY 2007. URL, http://www.ibm.com/developerworks/webservices/library/ws-rest1/, last viewed 2008-03-24.

[15] Chu, HoJin: *See the Power of Symbian OS (Introduction and Overview)*, AUG 2007. URL, www.software.or.kr/ICSFiles/afieldfile/2007/08/30/See_the_Power_of_Symbian_OS_KIPA.pdf, last viewed 2008-01-31.

[16] Costello, Roger L.: *Building Web Services the REST Way*. URL, http://www.xfront.com/REST-Web-Services.html, last viewed 2008-03-24.

[17] Erl, Thomas: *Service-Oriented Architecture: A Field Guide to Integrating XML and Web Services*. Prentice Hall International (6. Mai 2004), 2004.

[18] Giguere, Eric: *Service-Oriented Architecture and Java ME*. OCT 2006. URL, http://developers.sun.com/mobility/midp/ttips/soaintro/, last viewed 2008-03-11.

[19] Grey, Kevin: *AJAX on IE Mobile*. NOV 2005. URL, http://blogs.msdn.com/iemobile/archive/2005/11/15/493200.aspx, last viewed 2008-02-28.

[20] Hansen, Mark: *Basic SOA Using REST*. MAY 2007. URL, http://www.ddj.com/java/199300123;jsessionid=

JTR1XT5I2YZNMQSNDLRSKHSCJUNN2JVN?_requestid=9156, last viewed 2008-03-24.

[21] Harakawa, Takuya and Dr. Tomihisa Kamada: *Embedded AJAX*, 2007. URL, http://www.access-company.com/PDF/NetFront/Ajax%20white%20paper.pdf, last viewed 2008-02-26.

[22] Jakl, Andreas: *Symbian OS Overview*, NOV 2007.

[23] Jipping, Michael J.: *Smartphone Operating System Concepts with Symbian OS*. Symbian Press (WILEY), 2007.

[24] Jones, Robert: *Creating Web Content for Mobile Phone Browsers, Part 1*, JUN 2004. URL, http://www.oreillynet.com/pub/a/wireless/2004/02/06/mobile_browsing.html, last viewed 2008-02-14.

[25] Kanellos, Michael: *Big changes ahead for microprocessors*. NOV 2001. URL, http://www.news.com/Big-changes-ahead-for-microprocessors/2009-1001_3-275823.html, last viewed 2008-04-24.

[26] Knudsen, Jonathan: *MIDP Application Security 2: Understanding SSL and TLS*. OCT 2002. URL, http://developers.sun.com/mobility/midp/articles/security2/, last viewed 2007-12-28.

[27] Kreft, Klaus and Angelika Langer: *Java Multithread Support – Threadpools*. MAY 2005. URL, http://www.angelikalanger.com/Articles/JavaSpektrum/20.ThreadPools/20.ThreadPools.html, last viewed 2008-05-12.

[28] LiMo Foundation: *LiMo Foundation Platform Architecture - White Paper Version 1.0*. JAN 2007. URL, http://www.limofoundation.org/images/stories/pdf/limo_platform_arch.pdf, last viewed 2008-04-05.

[29] LiMo Foundation: *LiMo Foundation - Introduction, Overview & Market Positioning*. FEB 2008. URL, http://www.limofoundation.org/images/stories/pdf/limo-foundation-overview-feb2008.pdf, last viewed 2008-04-05.

[30] Mahmoud, Qusay H.: *Service-Oriented Architecture (SOA) and Web Services: The Road to Enterprise Application Integration (EAI)*. APR 2005. URL, http://java.sun.com/developer/technicalArticles/WebServices/soa/, last viewed 2008-03-11.

[31] Makofsky, Steve: *Pocket PC Network Programming*. Addison-Wesley Professional, 2003.

[32] Malone, Michael S.: *Moore's Second Law*. APR 2004. URL, http://www.wired.com/wired/archive/12.04/start.html?pg=2, last viewed 2008-04-24.

[33] McCarthy, Philip: *Ajax for Java developers: Build dynamic Java applications*. SEP 2005. URL, http://www.ibm.com/developerworks/library/j-ajax1/, last viewed 2008-02-25.

[34] McLaughlin, Brett: *Ajax: A New Approach to Web Applications*. FEB 2005. URL, http://adaptivepath.com/ideas/essays/archives/000385.php, last viewed 2008-02-25.

[35] McLaughlin, Brett: *Mastering Ajax, Part 1: Introduction to Ajax*. DEC 2005. URL, http://www.ibm.com/developerworks/web/library/wa-ajaxintro1.html, last viewed 2008-02-25.

[36] McLaughlin, Brett: *Mastering Ajax, Part 2: Make asynchronous requests with JavaScript and Ajax*. JAN 2006. URL, http://www.ibm.com/developerworks/web/library/wa-ajaxintro2/index.html, last viewed 2008-02-25.

[37] Microsoft Corporation: *Architectural Overview of Windows Mobile Infrastructure Components (Windows Mobile 5.0 and 6-powered devices)*, MAY 2007. URL, http://download.microsoft.com/download/c/b/d/cbdc18d1-1a01-4736-a557-08474ec73443/Windows_Mobile_Architecture_Overview.pdf, last viewed 2008-01-18.

[38] Mills, Chris: *JavaScript support in Opera Mini 4*. OCT 2007. URL, http://dev.opera.com/articles/view/javascript-support-in-opera-mini-4/, last viewed 2008-02-28.

[39] Mindbreeze Software GmbH: *Developers Guide Mindbreeze Enterprise Search 3.0 SDK*, 2007.

[40] Monson-Haefel, Richard: *J2EE Web Services*. Addison-Wesley Professional; 1st edition (October 17, 2003), 2003.

[41] Nokia Corporation: *maemo 4 Quick Start Guide*. OCT 2007. URL, http://maemo4beginners.garage.maemo.org/maemo-quick-start-guide.pdf, last viewed 2008-04-05.

[42] OpenAjax Alliance: *Mobile Device APIs*. Website, MAY 2008. URL, http://www.openajax.org/member/wiki/Mobile_Device_APIs, last viewed last viewed 2008-05-30.

[43] Opera Software ASA: *The Opera Platform*. URL, http://www.opera.com/products/mobile/brochures/OperaPlatform.pdf, last viewed last viewed 2008-05-30.

[44] Ortiz, C. Enrique: *Understanding the Web Services Subset API for Java ME*. MAR 2006. URL, http://developers.sun.com/mobility/midp/articles/webservices/, last viewed 2008-03-27.

[45] PalmSource, Inc: *Introduction to Palm OS Developer Suite*, 2004. URL, http://www.access-company.com/developers/documents/docs/dev_suite/PalmOSDevSuite/ToolsTOC.html, last viewed 2008-01-18.

[46] Passani, Luca: *WAP Forum - Wireless Application Protocol WAP 2.0 Technical White Paper*. URL, http://developer.openwave.com/dvl/support/documentation/guides_and_references/xhtml-mp_style_guide/index.htm, last viewed 2008-02-14.

[47] Pollington, David: *Web Runtimes – evolving beyond the browser*, FEB 2008. URL, http://mobilemonday.org.uk/Vodafone\%20MoMo\%20Feb\%204.ppt, last viewed 2008-05-30.

[48] Prescod, Paul: *REST and the Real World*. FEB 2002. URL, http://webservices.xml.com/pub/a/ws/2002/02/20/rest.html, last viewed 2008-03-24.

[49] Research In Motion Limited: *BlackBerry and Java*, MAR 2002. URL, http://www.blackberry.net/developers/javaknowledge/presentations/download/BlackBerry_JDE.pdf, last viewed 2008-01-11.

[50] Research In Motion Limited: *BlackBerry Java Development Environment - BlackBerry Application Developer Guide Volume 1: Fundamentals*, SEP 2006.

[51] Research In Motion Limited: *Developing Applications for BlackBerry Devices: An Introduction for Mobile Application Developers*, DEC 2006.

[52] Research In Motion Limited: *Developing Mobile Applications - Design Principles for BlackBerry Browser Applications*, APR 2006. URL, http://www.blackberry.com/knowledgecenterpublic/livelink.exe/Design_Principles_for_BlackBerry_Applications.pdf?func=doc.Fetch\&nodeId=1206298\&docTitle=Design+Principles+for+BlackBerry+Applications, last viewed 2008-01-11.

[53] Research In Motion Limited: *Introduction to the Plazmic Content Developer's Kit for Rich Media Applications*, 2006.

[54] Research In Motion Limited: *Research In Motion*, 2008. URL, http://www.blackberry.com/select/get_the_facts/pdfs/rim/RIM_Presentation_Q1_Fiscal_2008.ppt, last viewed 2008-01-11.

[55] Research In Motion Limited: *Research In Motion - History*, 2008. URL, http://www.blackberry.com/select/get_the_facts/pdfs/rim/rim_history.pdf, last viewed 2008-01-11.

[56] Richardson, Leonard and Sam Ruby: *RESTful Web Services*. O'Reilly Media, Inc. (May 8, 2007), 2007.

[57] Rui, Shu Fang: *Designing mobile Web services*. JAN 2006. URL, http://www.ibm.com/developerworks/wireless/library/wi-websvc/, last viewed 2008-05-23.

[58] Schroepfer, Mike: *Mozilla and Mobile*. OCT 2007. URL, http://weblogs.mozillazine.org/schrep/archives/2007/10/mozilla_and_mobile.html, last viewed 2008-02-29.

[59] Shearer, Findlay: *Power management in mobile devices – A view of energy Conservation*. APR 2008. URL, http://www.planetanalog.com/features/showArticle.jhtml?articleID=207100756, last viewed 2008-04-24.

[60] Smidt Hansen, Soren: *.Net Compact Framework Overview*, NOV 2002. URL, http://www.daimi.au.dk/~ups/DOPC/slides/12_compact.ppt, last viewed 2008-01-19.

[61] Souders, Steve: *High Performance Web Sites. 14 Steps to Faster-Loading Web Sites: Essential Knowledge for Front-end Engineers*. O'Reilly Media, Inc. (September 11, 2007), SEP 2007.

[62] Sun Microsystems, Inc.: *Java ME Platform Overview*. Website. URL, http://java.sun.com/javame/technology/index.jsp, last viewed 2007-12-26.

[63] Sun Microsystems, Inc.: *Connected, Limited Device Configuration - Specification Version 1.0a*, MAY 2000. PDF, CLDCspec10a.pdf.

[64] Sun Microsystems, Inc.: *J2ME Building Blocks for Mobile Devices - White Paper on KVM and the Connected, Limited Device Configuration (CLDC)*, MAY 2000. URL, http://java.sun.com/products/cldc/wp/KVMwp.pdf, last viewed 2007-12-28.

[65] Sun Microsystems, Inc.: *Mobile Information Decvice Profile Version 2.0*, NOV 2002. PDF, midp-2_0-fr-spec.pdf.

[66] Sun Microsystems, Inc.: *Connected, Limited Device Configuration - Specification Version 1.1*, MAR 2003. PDF, CLDCSpecification1.1.pdf.

[67] Sun Microsystems, Inc.: *Java Technology for the Wireless Industry (JSR-185) - Road Map 1 Definition*, JAN 2003. URL, http://java.sun.com/j2me/docs/j2me_wireless_industry.pdf, last viewed 2007-12-29.

[68] Sun Microsystems, Inc.: *CLDC HotSpot Implementation Virtual Machine*, FEB 2005. URL, http://java.sun.com/j2me/docs/pdf/CLDC-HI_whitepaper-February_2005.pdf, last viewed 2007-12-26.

[69] Sun Microsystems, Inc.: *Java ME Technology At-A-Glance*, MAY 2006. URL, http://www.sun.com/aboutsun/media/presskits/javaone2006/JavaMEtechnology_aag.pdf, last viewed 2008-05-28.

[70] Sun Microsystems, Inc.: *Mobile Service Architecture Specification - Version 1.00*, SEP 2006. PDF, JSR 248 Specification v1.00 (FR).pdf.

[71] Sun Microsystems, Inc.: *Sun Mobile Device Technology - Introduction to Mobility Java Technology*. Website, DEC 2007. URL, http://developers.sun.com/mobility/getstart/, last viewed 2007-12-26.

[72] Symbian Ltd.: *Symbian Market Round-Up – Issue 2, 2007*, NOV 2007. URL, http://developer.symbian.com/main/getstarted/newsletter/MarketRoundUp/SymbianMarketRound-UpIssue2Oct07FINAL.pdf, last viewed 2008-01-30.

[73] Theurer, Tenni and Wayne Shea: *Performance Research, Part 5: iPhone Cacheability - Making it Stick*. FEB 2008. URL, http://yuiblog.com/blog/2008/02/06/iphone-cacheability/, last viewed 2008-02-27.

[74] Tilkov, Stefan: *Interview with Sanjiva Weerawarana: Debunking REST/WS-* Myths*. FEB 2007. URL, http://www.infoq.com/articles/sanjiva-rest-myths, last viewed 2008-03-24.

[75] Tyagi, Sameer: *RESTful Web Services*. AUG 2006. URL, http://java.sun.com/developer/technicalArticles/WebServices/restful/, last viewed 2008-03-24.

[76] WAP Forum: *Wireless Application Protocol - Wireless Telephony Application Interface Specification*, NOV 1999. URL, polylab.sfu.ca/spacesystems/teach/wireless/wap/documents/SPEC-WTAI-19991108.pdf, last viewed 2008-02-14.

[77] WAP Forum: *WAP Forum - Wireless Application Protocol White Paper*, JUN 2000. URL, www.wapforum.org/what/WAP_white_pages.pdf, last viewed 2008-02-14.

[78] WAP Forum: *WAP CSS Specification*, OCT 2001. URL, http://www.wapforum.org/tech/documents/WAP-239-WCSS-20011026-a.pdf, last viewed 2008-02-16.

[79] WAP Forum: *XHTML Mobile Profile*, OCT 2001. URL, http://www.openmobilealliance.org/tech/affiliates/wap/wap-277-xhtmlmp-20011029-a.pdf, last viewed 2008-02-15.

[80] WAP Forum: *WAP Forum - Wireless Application Protocol WAP 2.0 Technical White Paper*, JAN 2002. URL, www.wapforum.org/what/WAPWhite_Paper1.pdf, last viewed 2008-02-14.

[81] Weerawarana, Sanjiva, Francisco Curbera, Frank Leymann, Tony Storey and Donald F. Ferguson: *Web Services Platform Architecture*. Prentice Hall PTR (April 1, 2005), 2005.

[82] Wilson, Greg, Jean Ostrem and Christopher Bey: *Palm OS Programmer's Companion Volume1*, NOV 2004. URL, http://www.access-company.com/developers/documents/docs/palmos/PalmOSCompanion/CompanionTOC.html, last viewed 2008-01-17.

[83] Wischy, Markus Alexander: *Erste Schritte mit dem .NET Compact Framework*. NOV 2004. URL, http://msdn2.microsoft.com/de-de/library/bb978945.aspx, last viewed 2008-03-27.

[84] World Wide Web Consortium: *Mobile Web Best Practices 1.0*, NOV 2006. URL, http://www.w3.org/TR/mobile-bp/, last viewed 2008-02-26.

[85] World Wide Web Consortium: *The XMLHttpRequest Object*, OCT 2007. URL, http://www.w3.org/TR/XMLHttpRequest/, last viewed 2008-02-26.

[86] Zakas, Nicholas C., Jeremy McPeak and Joe Fawcett: *Professional Ajax, 2nd Edition*. WROX Press Ltd., MAR 2007.

[87] Zheng, Pei and Lionel M. Ni: *SMART PHONE & NEXT GENERATION MOBILE COMPUTING*. Morgan Kaufmann, 2006.

Curriculum Vitae

Christian P. Praher

Zur Person

Name	Christian Paul Praher
Anschrift	Blütenstraße 23/11/68, A-4040 Linz
Telefon	+43 664 / 4500 433 (tele.ring)
E-Mail	chris.praher@gmx.net
Geburtsdatum	15. Februar 1982 (Linz)
Familienstand	Ledig
Staatsbürgerschaft	Österreich
Führerscheinklassen	A, B

Ausbildung

Grundschule	
1988 - 1992	Volksschule Aigen-Schlägl
Unterstufe	
1992 - 1996	Bundesgynmasium Rohrbach
Oberstufe	
1996 - 2001	Handelsakademie Rohrbach, abgeschlossen mit Matura
Präsenzdienst	
2001 - 2002	Zivildienst Bezirksaltenheim Hart-Leonding
Studium	
2002 - 2006	Bakkalaureatsstudium Informatik an der Johannes Kepler Universität Linz, abgeschlossen mit Bakk.tech.
Seit 2006	Masterstudium Informatik an der Johannes Kepler Universität Linz
2/2007 - 6/2007	Auslandssemester an der Université de Marne-La-Vallée (Paris/Frankreich)

Berufliche Aktivitäten

1999 - 2008

Praher KEG

Realisierung von Projektarbeiten zusammen mit meinem Bruder, DI Jakob Praher, vor allem im Bereich Webapplikationen (JavaEE) und Netzwerke.

2002 - 2008

SBX-Mathematik

Entwicklung eines E-Learning Frameworks und entsprechenden Beispielen auf Basis von Microsoft Active Server Pages für die Schulbuch eXtra Inhalte (SBX) der Bücher „Mathematik I - Mathematik V“ des Trauner Verlages.

2004 - 2005

IT-Math

Implementierung von Java-Applets für die Visualisierung mathematischer Algorithmen im Rahmen des Projektes „IT-Math“ am Institut für Informationsverarbeitung und Mikroprozessortechnik (FIM) der Johannes Kepler Universität Linz.

Seit 4/2008

FIM

Wissenschaftlicher Mitarbeiter ohne Diplom am Institut für Informationsverarbeitung und Mikroprozessortechnik (FIM) der Johannes Kepler Universität Linz.

Eidesstattliche Erklärung

Hiermit erkläre ich an Eides statt, dass ich die vorliegende Arbeit selbstständig und ohne fremde Hilfe verfasst, andere als die angegebenen Quellen und Hilfsmittel nicht benutzt und die aus anderen Quellen entnommenen Stellen als solche gekennzeichnet habe.

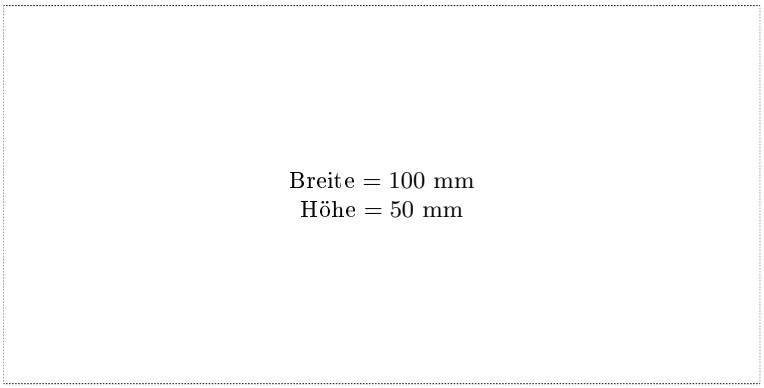
Des weiteren versichere ich, dass ich diese Masterarbeit weder im In- noch im Ausland in irgendeiner Form als Prüfungsarbeit vorgelegt habe.

Linz, am 4. Juni 2008

Christian P. Praher

Messbox zur Druckkontrolle

— Druckgröße kontrollieren! —



— Diese Seite nach dem Druck entfernen! —