TECHNISCHE UNIVERSITÄT WIEN

VIENNA UNIVERSITY OF TECHNOLOGY

MASTERARBEIT

A Mobile Blogging Solution - Aggregation and Management of location-based Content in the Java Application Domain

Ausgeführt am Institut für
Softwaretechnik und interaktive Systeme (ISIS)
der Technischen Universität Wien

unter der Anleitung von
Univ.-Prof. Dipl.-Ing. Dr.techn. Hannes Werthner
und Mitbetreuung von
Univ.-Ass. Mag. Christoph Grün
Projektass. Mag. Petra Brosch

durch
Hannes Weingartner
Friedmanngasse 51/11
1160 Wien

Wien, 23. April 2008 Unterschrift

Eidesstattliche Erklärung

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

Wien, 23. April 2008 Unterschrift

Danksagung

Die vorliegende Arbeit wäre ohne die Unterstützung von zahlreichen Personen wohl in dieser Form nicht zustande gekommen. Ein besonderer Dank gilt Mag. Christoph Grün von der *Electronic Commerce Group* (EC) und Mag. Petra Brosch von der *Business Informatics Group* (BIG), welche mich beide von Anfang an hervorragend bei meinem Unterfangen unterstützt haben. Ich bedanke mich ebenfalls bei Univ.-Prof. Hannes Werthner welcher die Betreuung für meine Abschlussarbeit übernahm. Weiters bot mir die *Interactive Media Systems Group* (IMS) die notwendige Ausbildung, aber auch die technischen Mittel um das Projekt realisieren zu können. Hier sei besonders Dr. Horst Eidenberger erwähnt, welcher mein Interesse an *M-Commerce* Systemen geweckt hatte. Allgemein ermöglichte mir die *Technische Universität Wien* und im Speziellen das *Institut für Softwaretechnik und Interaktive System* (ISIS), als auch die *Technische Universität Helsinki* und die *Technische Universität Kiew* eine fundierte Ausbildung zum Diplom-Ingenieur im Bereich der Informatik.

Für eine Unterstützung und ein Interesse an meiner Arbeit in außeruniversitären Kreisen bedanke ich mich beim M-Commerce Softwareerzeuger *IP Square* und besonders bei Clemens Mensik. Meine Eltern und meine Schwester Bettina gaben mir außerdem Anerkennung und Motivation bei meinem Vorhaben in den letzten Jahren, welche schließlich auch zum erfolgreichen Abschluss meiner Ausbildung beitrugen. Aus meinem Freundeskreis sei besonders Patrik Wielath erwähnt, welcher mich in technischen Belangen als auch bei der Testphase unterstützte. Für ein Korrekturlesen meiner Arbeit bin ich Andrea Trautsamwieser, Mag. Ulrike Öhlinger und Mag. Bettina Nenning zu Dank verpflichtet. Auch Tommi Laukkanen, dessen Arbeit *TrailExplorer* mich sehr inspirierte, und weitere Software Ingenieure aus diversen Online Foren boten mir zahlreiche Tipps während der Implementierungsarbeit.

Kurzfassung

Mobilgeräte wie etwa *Smart Phones* sind für gewöhnlich permanente Begleiter im Alltag. Sie bieten per se ein flexibles Kommunikationsmedium in unserer Gesellschaft, wobei zahlreiche Dienste die herkömmliche Sprachtelefonie ergänzen. Der Daten-Durchsatz und die Anbindung an das mobile Internet durch *Zellulare Netzwerke* und *Öffentliche Hotspots* steigt stetig an. Ubiquitäre Mobilität hat heutzutage eine hohe Signifikanz für Geschäftsprozesse und Endverbraucher-Anwendungen. Die Verarbeitung von dynamischen Umgebungs-Parametern, wie etwa die geodätische Position eines Mobilgerätes, ermöglicht in diesem Zusammenhang eine Vielzahl von benutzerbezogenen Diensten. Eine Bereitstellung dieser Dienste über zeitliche und räumliche Grenzen hinweg erlaubt zudem eine flexible Interaktion mit beliebigen Daten durch Einbeziehung mehrerer Netzwerke.

Die Diplomarbeit liefert anfänglich einen technischen Wegweiser um den Eigenheiten von mobilen Plattformen, drahtlosen Netzwerk-Infrastrukturen und Positionierungs-Techniken gewahr zu werden. Das Ziel ist eine effiziente Ausnutzung von *Operator*- als auch *Internet*- und *Geräte*-basierte Diensten auf wenig leistungsfähigen Mobilgeräten. Zudem werden abschätzbare technologische Erweiterungen für die Realisierung von *Pervasive Computing* in zukünftigen Anwendungen diskutiert. In weiterer Folge werden technische Fragen bezüglich einer Anwendungsarchitektur und einer Implementierung umrissen, welche auf einem funktionalen Prototypen basieren. Der Prototyp realisiert eine *Mobile Blogging Anwendung*. Die Anwendung representiert ein verteiltes Java-basiertes Software System, welches mobile als auch statische Komponenten beinhaltet. Blogs werden auf Mobilgeräten erzeugt und bestehen aus einem *Foto*, einer *Nachricht*, als auch aus Informationen zum *Standort* und *Zeit* der Blog-Generierung. Zusätzliche Informationen wie etwa *Höhenangaben*, *logische Standortinformation*, und zurückgelegte *Distanzen* sind zudem einfach abrufbar. Diese geografisch ettiketierten Blogs sind für eine nachfolgende Manipulation und Exploration auf dem Mobilgerät verfügbar. Ein *Blogging Dienst* gibt dem mobilen Benutzer die Möglichkeit diese benutzerzentrierten Blogs auf einen *Anwendungsserver* zu transferieren. Die Blogs sind dadurch über eine web-basierte *kartographische Schnittstelle* zugänglich. Der Zugriff auf erstellte und gepostete Blogs liegt in vollkommener Kontrolle des Urhebers. Die Annotationen in einer virtuellen Landkarte durch die zu den Blogs gehörigen *Wegpunkte* ermöglichen eine aussagekräftige Darstellung über bestimmte Aktivitäten. Die Anwendungsfelder liegen demnach hauptsächlich im Tourismus-Sektor aber auch in der Forschung, wo eine standortbezogene *Dokumentation* von Artefakten als sinnvoll erscheint.

Viele *Frameworks* und *Tools* sind für den mobilen Anwendungsentwicklungs-Prozess verfügbar um die Standardfunktionalität auf mobile Plattformen zu erweitern. In der prototypischen Implementierung führt die Bereitstellung eines solchen Frameworks zu einer höheren Benutzerfreundlichkeit und zu einer verbesserten Code-Wartbarkeit als auch Programmfehlerbeseitigung.

Abstract

Mobile handhelds like *Smart Phones* are usually permanent attendants in everyday life. They offer per se flexible means of communication in our society, whereby many services are complementing the conventional voice telephony. The data throughput and the accessibility of the mobile Internet through *Cellular Networks* and *Public Hotspots* is steadily increasing. Ubiquitous mobility has a high significance today, both for business processes and consumer applications. The processing of dynamic environmental parameters like the device's geodetic position enables in this context a variety of user-aware services. Furthermore, a deployment of services across temporal and spatial bounderies allows a flexible interaction with arbitrary data by incorporating multiple networks.

This master's thesis initially supplies a technical roadmap to get aware of the peculiarities of mobile platforms, wireless network infrastructures and positioning techniques. The aim is an efficient utilization of *Operator*- as well as *Internet*- and *Device*-based services on resource-limited mobile devices. Furthermore, foreseeable technological enhancements for realizing *Pervasive Computing* in future setups are discussed. Subsequently, technical issues regarding application architecture and implementation are outlined based on a working prototype. The prototype realizes a *Mobile Blogging Application*. This application represents a distributed Java-based software system including mobile and static components. Blogs are created on handhelds and consist of a *Photo*, a *Message* as well as information about *Location* and *Time* of the blog creation. Additional information like *Elevation Levels*, *logical Location Information*, and covered *Distances* can be obtained easily. These geotagged blogs are available for a subsequent manipulation and exploration in a mobile context. A *Blogging Service* gives the mobile user the opportunity to transfer these user-centric blogs to an *Application Server*. The blogs are thereby accessable through a web-based *cartographic Interface*. The access to the created and posted blogs is under full control of the content owner. The annotations within a virtual map through blog-related *Waypoints* enable an expressive representation of distinct activities. The application scenarios are thus mainly in the tourism sector, but also in the research sector in cases where a location-based *Documentation* of artifacts seems to be useful.

Many *Frameworks* and *Tools* are available for the mobile application developing process to extend the standard functionality of platforms. In the prototype implementation, a deployment of such a framework increases the usability and improves the code maintainability as well as the debugging process.

Contents

1. Introduction

"What on earth would a man do with himself,
if something didn't stand in his way?"

— H.G. Wells (1866-1946)

1.1. Presentation of the Problem

Ubiquitous mobility becomes more and more important in the information technology sector. The introduction of enterprise paradigms like service-oriented computing for a flexible interaction with remote services across platforms has changed the access to information in the last years. Handheld devices provide a very flexible way for accessing such services and according resources from nearly every place in the world. Next to the standard communication services, mobile platforms and telecom infrastructures offer enough resources to incorporate sophisticated multimedia features nowadays, such as high resolution photo capturing and video streaming. Furthermore, operator facilities and local *Global Positioning System (GPS)* receivers are suitable to determine one's location, which might be an essential parameter for ubiquitous applications. In this relationship it is important to ensure the user's privacy and to restrict access to sensitive data among several domains.

The delivery of mobile content accross multiple networks in form of business transactions or social communication is a commonly performed task in mobile infrastructures. *Mobile Blogging Applications* are suitable for managing and posting multimedia data like text messages in combination with photos or audio and video content. Such applications are on the way to become attractive social utilities for mobile end users. However, limited resources, highly dynamic network connections and heterogeneous feature implementations on mobile platforms are technology intrinsic and unavoidable facts today. Those aspects have to be considered when dealing with handheld-devices, also in terms of multimedia data capturing and processing. As a consequence, developing advanced and reliable mobile applications is often a challenging task. Furthermore, remote services rely on different communication models, whereby a service client has to understand the according protocols and the involved data types. Since most web services do not consider these facts, only a small amount of them is useful in the mobile domain. However, additionally to standardized mobile platform implementations, several mobile frameworks and tools with enhanced and useful features are available nowadays. They might help a software developer to create novel and widely executable applications for handheld devices.

The *General Packet Radio Service (GPRS)* and related services as well as *Third Generation (3G)* cellular networks are today the standard bridging technologies for exchanging data with the wired network. These mobile infrastructures provide a maximal device portability. Wireless link instability and server failures, however, may cause a service interruption and sometimes even an

abortion of an active data transfer and therefore undesirable costs for the end user. Other built-in radio-based technologies like *Wireless Local Area Network (WLAN)* or *Bluetooth* (BT) provide a service access with a high reliability and data throughput at no data transfer cost. The main attention in this relationship lies in the restricted mobility, which is intrinsic for those network technologies.

1.2. Motivation

One goal in this master's thesis is to convey an awareness and a utilization of services, which are designed to work within a mobile context as well as those, which have their roots in web-based environments. The theoretical part focuses on these aspects and describes properties of mobile service environments, including wireless infrastructures, positioning techniques, and distributed middleware systems. Services are flexible self-contained components and therefore suitable to access heterogenous distributed information and processes. The availability of corresponding on-device parsing and service invocation features are essential requirements for such a straightforward access. Based on such features a *Mobile Blogging Application*, called *TMBlog*, for Java-related platforms is developed in the course of this master's thesis. The according prototype implementation allows a flexible location-aware blog generation process through the utilization of on-device features, short-range external services, operator facilities, and web-residing services. The application aims to offer a platform for exploring and managing these gathered blog content parameters in a suitable way.

A mobile service endpoint offers a *Gateway* to the wired Internet, in order to map selected user-centric blogs into a web context. Due to current restrictions on mobile platforms, some thoughts are necessary to maintain a suitable service-oriented payload exchange for supplying binary assets as well. The according application server enables user registration-, user authentication-, and persistence features, to enable a secure device-aware *Identity- and Session Management* as well as a suitable *Blog Access Management*. In analogy to ordinary greeting cards, sent by people from far away places, delivered mobile blogs may also provide user-centric data like a *Photo*, a message body and a postmark in form of *Location Coordinates* as well as a timestamp of creation. Consequently, these geotagged blogs are waypoints which are temporal and spatial related among each other, suitable for an exploration within a cartographic web-based interface through a known *Community*. Based on these waypoints a personalized mobile *Tracking* feature is an obvious extension, to allow a subsequent insight into one's personal *Trace* already on the mobile handset. Additional parameters like *logical Location Information*, *Elevation Levels*, and covered *Distances* offer the mobile user further information about visited places. An adequate application design allows a batch-based processing of created blogs, which are part of a distinct journey named by the user. This includes an automated content transfer of several blogs at once. A list of posted blogs and the volume of transfered data are also available features of the prototype. The web client logic automatically loads blogs from the application server in near real-time, as long as they are posted during an active web session. Furthermore, Java-based frameworks are aimed to be utilized to improve the user's experience through advanced mobile user interface components. In all domains a sophisticated software development process in *Model-View-Control (MVC)*-style and on-device logging functionalities are provided.

Figure 1.1 refers to the mentioned and additionally desired components of the prototype residing in multiple domains. The upper part of the illustration involves the mobile components, the lower part refers to the static components.

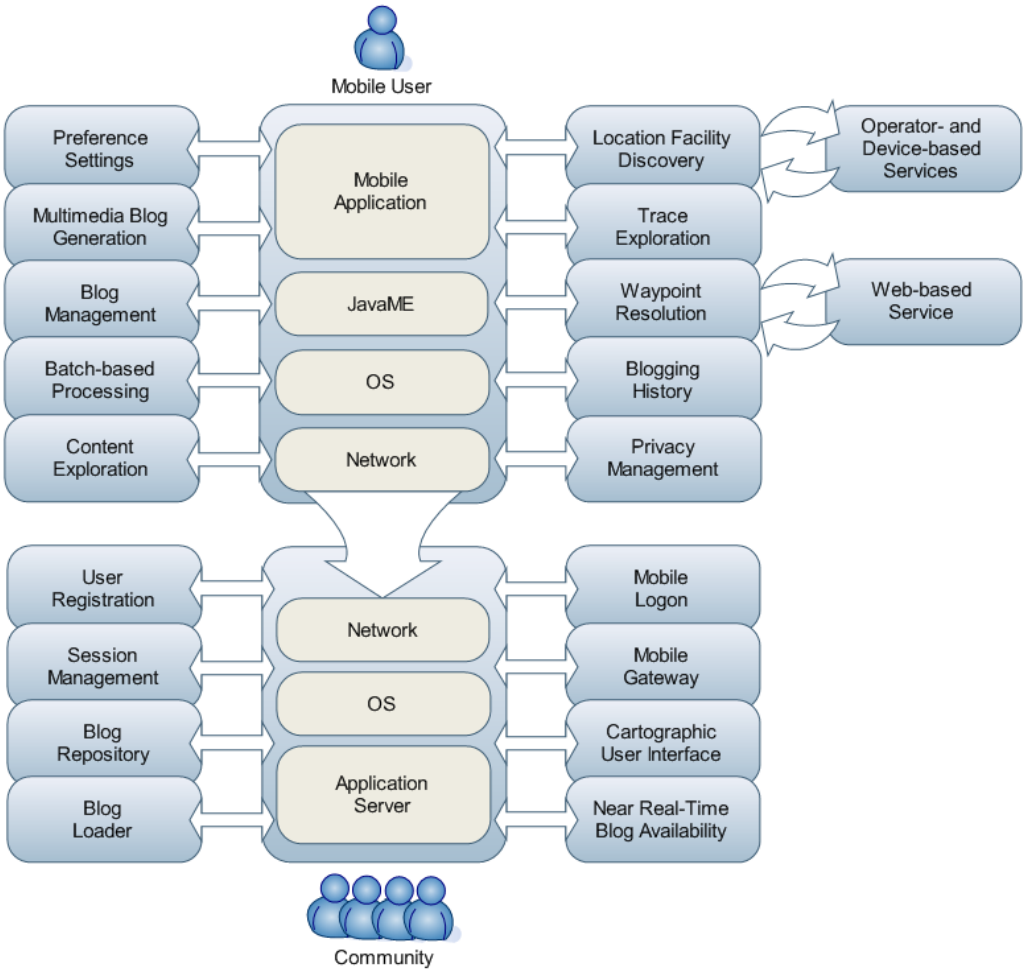


Figure 1.1.: *Schemata of the Prototype with all its Components, distributed among different Domains. A location-based Service and a Blogging Service are accessable through the Mobile Client. The Mobile- as well as the Static Platform Components are executed within a Java-based Environment.*

Before the actual application is *designed* and *implemented*, it is important to understand the contextual *Background* and the *Related Technologies*. An outline of the chapters of this master's thesis is offered in the next section.

1.3. Outline

Chapter *Background* [2] briefly introduces the current situation of the *Mobile IT Market* and its influence on the *Tourism Sector*. Furthermore, properties of *Mobile Blogging* as well as those of a user's *Privacy* in relationship to social and technical aspects are described. In addition, opportunities related to *Ubiquitous Mobility* and *Service-Oriented* interaction models in the mobile domain are motivated.

Chapter *Analysis of Related Technologies* [3] is an extensive chapter about *Mobile Platforms* and *Wireless Facilities* as well as about *Computing Paradigms* and *Location Determination* techniques. The aim of this roadmap is to get an awareness of hardware- and software-based service infrastructures in order to understand software architectures and -constraints for the utilization of mobilized services on handhelds.

Chapter *Requirements* [4] offers *Tourism-related Studies* in the mobile IT industry, possible *Application Scenarios* and the desirable interactions of the user with the *Blogging System*. The analysed *User Interaction Cases* are directly related to the discussed *Functional Requirements*. The functionality is distributed over several domains, however the main focus relies on the *Mobile Client Domain* and its *Control Flow* schemata. These schemata enable an analysis of component functions and -interactions for performing essential tasks on the mobile platform.

Chapter *Prototype Design* [5] determines the *Architecture* of the application on different levels of abstraction. The actual structure of the *Mobile Core System* and the *Server-side Components* is outlined in this chapter. The functionalities as they have been defined in chapter *Requirements* are mainly affecting the underlying design decisions. The mobilized logical core components also refer to concrete *Java Classes* and their APIs. Furthermore, this phase offers inter-domain interaction models to specify the *Temporal Behaviour* between local and remote functionality, as well as time-critical operations.

Chapter *Prototype Implementation* [6] provides *Code Outlines* from multiple domains. The focus lies on the application's *Service Modules*, the *Content Generation Management*, and the *Content Access Management* as well as on the according *Class Diagrams*. The chapter also discusses the application's *Interfaces* in the mobile- and the web domain.

Chapter *Related Work and Evaluation* [7] presents *Related Academic and Commercial Works* in the mobile IT industry. Furthermore this chapter summarizes the *Experience* which has been made during the prototype implementation and testing phases also in correlation with the covered knowledge through prior studies. A discussion about occured *Problems* and an *Evaluation* of the prototype in comparision with the mentioned related projects give a conclusion about the quality of the developed approach.

Chapter *Conclusion* [8] summarizes the key factors of this master's thesis. Discussed *Perspectives* on upcoming technologies in the mobile domain completes the master's thesis.

2. Background

"Computers are incredibly fast, accurate, and stupid;
humans are incredibly slow, inaccurate and brilliant;
together they are powerful beyond imagination."

— Albert Einstein (1879-1955)

2.1. Mobile Information Technology Market

The first smart phones and communicators became available on a crumbling monopolistic telecom market in the late 90s, while the development of customized mobile solutions got initiated [Sil07]. Since this time, mobile devices and carrier-based services are an interesting platform for individual software developers as well as for companies to expand their business infrastructures in correlation with privately owned technologies. The enterprise is an important player in this context, whereby about 28% made use of cellular data networks in Q4 2007 [Sil07]. Enterprises steadily develop their mobile platforms by introducing new hardware and software concepts, whereby this process is mainly driven by interests of companies, organizations and consumer studies.

In Q2 2007 mobile device shipments in the EMEA[1] region are distributed as follows[2]: *Nokia* (51%), *Samsung* (17%), *Sony Ericsson* (14%), *Motorola* (6%), Others (12%). Worldwide about 290 million units have been shipped per quartal in 2007. As the dominant leader within the european market, Nokia provides the following core products: the all-round *N95* (S60), the business-oriented *E65* (S60), and the mass market *6300* (Series 40) handsets. All these core models include a high resolution camera, as well as Java APIs for supporting *Mobile Media*, *Web Services* and *Bluetooth* by default. The multimedia-based *NSeries* models and the business models also include the *Location* API. Since the listed optional Java APIs are supported by other companies in a similar way and a pure Java application is not bound to a specific seller or user interface platform and as long as an *Java Virtual Machine (JVM)* is available on the device, it potentially reaches a deployment on the highest possible user base. However, there are some gaps which might prevent a successful deployment on some devices or even groups of devices (see section *Code Efficiency* [3.2.1]).

[1] Western Europe, Central and Eastern Europe, Middle East, Africa
[2] http://www.idc.com

2.2. Tourism and Information Technology

Due to the vast number of mobile handheld devices and their agility of accessing IT infrastructures, mobile applications already play a crutial role for a dynamic society. The tourism sector is an important application domain in this relationship, which already utilized web infrastructures in the mid-90's. According to Werthner and Klein [WK99], between 33% and 50% of the web transactions were somehow related to tourism at this time. Furthermore, *Computerized Reservation Systems* (CRS) and *Global Distribution Systems* (GDS) are still central intermediary components between customers and service suppliers in this sector. Actors like *Travel Agents*, *Tour Operators*, *Transportation Agencies*, and *Hotel Chains* are linked together through vertical networked-based communication strategies, whereby services are integrated into others, belonging to the same transaction chain.

Modern tourism-based systems are more related to additional services and a better as well as a mobile integration of information infrastructures for end users in a price sensitive and less loyal way [WK99]. However one has to consider, that this domain might accept distinct IT solutions only for short periods until they are outdated and subsequent trends are aimed to be followed. The establishment of context-aware and ubiquitous systems are along novel strategies, which are responsible for a structural change towards modern and possibly long-lasting systems, strongly related to the *Mobile Commerce* sector. *Location-based Services* (LBSs) are instances of context-aware systems with an increasing importance for building advanced mobile applications. Since information retrieval and social interactions are driven aspects in tourism-based settings, tracing and publishing experiences related to foreign areas in form of location-based content are typical actions performed by a tourist.

2.3. Blogging and Social Aspects

User-generated and published digital content in form of photos, videos, audios or simple text posted to the world wide web is also known as *Weblog* or *Blog*. Since blogs are created by millions of web- as well as by mobile users with variable interests, a huge pool of content also referred to as *Citizen Media* is online. An intermediate content aggregator has become obsolete. People usually communicate with each other to express themselves as a part of their social life. They provide options on news, share travel hints, memorable photos, and further content and experiences. Through blogs, which are acting as self-contained and public content carriers, they aim at reaching either people they already know or as many interested web users as possible, sometimes with the background to get to know like-minded people. This kind of communication is highly dynamic and the carried information reaches potentially as many people as people having access to the Internet, which obligatory results to the formation of many linked live communities.

2.3.1. World Live Web

Technorati [3] is a recognized authority, which made the monitoring of the so called *World Live Web* to its business. This authority is tracking for instance links between several hundred million

[3] http://www.technorati.com

blogs and social media. It also provides statistics based on the collected data. According to Technorati, there are about 175.000 new blogs and 1.6 million blog updates every day, which means about 18 updates per second. Many frameworks and tools like *Apache Roller* are currently available to develop corresponding platforms and portals for posting content in form of this *Web 2.0* related blogging concept.

2.3.2. Mobile Blogging

Since the mobile communication market as well as the performance and multimedia capabilities of corresponding devices are steady growing, *Mobile Blogging* is getting more attractive. Moreover, user-generated content on handhelds and the presence of wireless data networks gives the user maximal flexibility in terms of *Mobility* and *Spontaneity*, as well as an active role as content provider for the world wide web. Many mobile blogging platforms based on a variety of technologies are available today and also Nokia supports this trend with its own products *Lifeblog*[4] and *Location Tagger*[5]. Built-in cameras deliver high resolution JPG and PNG photos or MPEG videos with qualities suitable for creating expressive and rich media content for blogs. Therefore, the presence of mobile device cameras has a major impact on the growth of social networks. Corresponding platforms are mainly focused on managing photo and video content, due to much more comfortable capturing opportunities compared to typing characters on multi-function keyboards, which are intrinsic to handhelds. However, one drawback of this behaviour is the higher network utilization when transfering large binary data to remote nodes. Application performance problems, based on an unreliable connection to cellular networks with limited bandwidth and high service charges usually put the user in a bad mood.

Telecom operators usually offer customized flat rate models to their customers, which cover a fixed amount of data capacity per month, reaching from a few mega bytes up to several giga bytes. The standard data rates for common cellular data networks are as follows[6]: *GPRS* (84.4 kbit/s), *EDGE* (220 kbit/s), and *UMTS* (384 kbit/s). These rates have only a technology-related meaning because the delivery is usually constained and the service availability also heavily depend on the user's location.

2.4. Location Awareness

Tracking device locations with high accuracy up to a few meters, accessing stored personal data like contact information and e-mails, or even the management of the whole mobile platform from remote processes have become realistic scenarios and have already been implemented in numerous services. This kind of user tracking and information handling offers many opportunities to develop advanced services for customers. However, one has to distinguish if there is a willigness to provide personal information or if this kind of information is somehow accessable to others, without a permission of the affected person. In both cases suitable security features play a crucial role. In the latter case, mobile platform security concepts like *Capabilities* have to be effective, in the way of notifying users when untrusted applications try to access sensitive platform features (see section *Native Security Concepts* [3.1.2.4] and *Java Service Platform*

[4] http://www.nokia.com/lifeblog/
[5] http://www.nokia.com/betalabs/locationtagger
[6] http://www.t-mobile.at

[3.4.7]). However this fact can certainly not prevent unwanted data access at a whole, due to implementation failures and lacks in the communication chain. Even successive recording of mobile user behaviour through governmental organizations in a more or less excessive manner cannot be ruled out.

2.4.1. Location-based Services

As a high potential application field, *Location-Based Services (LBS)* often process frequently changing location information automatically, with the aim to track mobile users to help them under special conditions. Such applications, which are often based on a bilateral agreement between the user and a wireless service provider might be useful in terms of establishing advanced services with a high user experience. For instance, a provider may offer a location-based information system as part of an interactive tourist guide or a distributed collaborative game. The most common way to determine the user location outdoors, is to use the globally available GPS satellite system. One must have a passive GPS device to establish a satellite link and to receive location information (see section *Satellite-based Positioning Systems* [3.5.2]). Stand-alone GPS receivers and whole navigation systems are available at moderate prices and the according hardware becomes more and more a standard feature on smart phones nowadays. According to an IDC[7] study in 2006, the LBS revenue is expected to grow from a few million dollars in 2005 to over 3 billion[8] dollars until 2010. This study affected the US market, but a steady LBS deployment might be seen as a global trend.

2.4.2. Location-based Privacy

As mentioned by Choi [CC07], information privacy is *'the ability of the individual to personally control information about one's self'*. There are some few empirical studies on LBS privacy as well. Choi references to a popular characterization of personal information in cyberspace from 1998, which are as follows: *Authorship Information* (e.g. telephone conversation, e-mail), *Descriptive Information* (e.g. birth date, gender, membership), and *Instrumental Mapping Information* (e.g. social security number, password). Location-based information however *can not* be classified in the above static categories, since it is usually a temporary and changing information related to a person. Due to this fact, it is obvious to distinguish between *static* and *dynamic* personal information. Dynamic or contextual data like a person's location may be an indicator of one's activities and identities of nearby people, which is clearly worth to be protected in some kind. Furthermore, according to [CC07] it is important to be aware that *'privacy concerns relates strongly to the degree of perceived psychological feelings when we lost the control power about personal information'*. So, what is the intention of people when publishing personal location information to a world-wide accessable medium and thereby running the risk of personal data abuse?

[7] http://www.idc.com
[8] milliard under european terms

2.4.3. Intentions of Users

In the last section it was mentioned that a person's location is a sensitive form of data. Location-based services must therefore shield this kind of data from unauthorized people, in a way which usually depends on the nature of the application and the agreement of the service customers.

A blogging application, which has a public characteristic by default, can also be seen as a way of accessing and managing personal data by the blogger *exclusively*. All that without allowing others to access and interact with this information. This strong privacy can be easily realized through a web-based login mechanism. However sharing memories and experience with the family and friends is a basic need of humans, strongly manifested in social interaction. Digital media may utilize multiple communication channels to interact with people in real-time, whereby the content might be prepared and consumed in various ways. A map-based communication interface may enhance the *User Experience* by virtually displaying a journey undertaken in the real world, in combination with specific waypoints enriched with multimedia content. Through this kind of visualization, a person who has not been on a particular journey may get more *immersed* in one's experience and stirring location-related events. Cartographic web services like Google Maps further allow a *Rich User Interaction*, by offering arbitrary zoom levels and customized map visualizations from every point in the world with variable resolutions. More information about LBS and positioning technologies can be found in section *Location-based Infrastructures* [3.5].

One important critera for the acceptence of a service by customers and therefore its success, is its trustworthiness. Even if 100% system security will be never possible, features like a strictly controlled *Identity-based Data Management* as well as strong *Encryped Sensitive Data* are the basis for highly reliable and trustable business and social applications. One should always have in mind, that applications in which the customer's privacy gets lost are not suitable for the market, regardless of the remaining product quality.

2.5. Ubiquitous Mobility

Mobile devices are nowadays usually permanent attendants, since accessability to other people and information systems became an important part within business and social activities. Connecting remote data and services from every location and in a 24/7 manner is an enduring quest within the enterprise IT industry. This liberty offers people many new opportunities and high flexibility. However, area-wide correlating and interference-free wireless networks and a high availability of physical access points in municipal areas are necessary technical infrastructures to provide a seamless interconnectivity across heterogeneous platforms. This fact does not correspond to our reality.

2.5.1. Degree of Mobility

[Per06] classifies mobility in three categories. In this sense the term *Fixed* indicates no mobility at all. In contrast, *Nomadic* users access a system from different places with different devices, whereas the location does not change during the interaction. *Mobile* users are also in movement while using a service. In the last category the service quality typically depends on the user's

environment and speed. Furthermore, the evolution of enterprise mobility can be separated in three phases [BTQ07]. In *'Phase 1: Islands of Connectivity'*, *Wi-Fi*[9] is deployed in collaborative areas with limited connectivity. *'Phase 2: Pervasive Wireless'* represents an established Wi-Fi network in a more cooperating manner like on a campus, by possibly augmenting some segments with GPRS cellular data cards. Finally, *'Phase 3: Ubiquitous Mobile Networks'* deploys the *WiMAX*[10] standard, whereby these municipal networks are seamlessly integrated into cooperating Wi-Fi networks. Phase 3 could become reality in about five years [BTQ07].

2.5.2. Pervasive Computing

Ubiquitous mobility might be the final stage in the evolution of the mobile IT infrastructure. A reliable connectivity between cellular data networks and *Wi-Fi* standards or *WiMAX*, has the potential to provide a seamless interaction of private and public information systems, widely accessable inside and outside major urban areas. According to [Sil07] *'Next generation carrier-based services must reach parity with 802.11n in throughput and increase in availability while decreasing in price'*. *Fourth Generation (4G)* networks and according services like WiMAX have the potential to let ubiquitous mobility become a realistic scenario. Nowadays several smart phones already have built-in WLAN capabilities, which might be used as an alternative to GPRS and *Third Generation (3G)* networks, whenever a connectivity to a local area network can be established. As indicated in the last section, even if WLAN and WiMAX access points and according device built-in capabilities will be highly available, only a smooth integration into country-wide carrier networks enables real *Pervasive Computing* —anytime, anywhere, any device [DD06].

2.5.3. Mobile Internet

In 2006 the term *Mobile 2.0* was first used to manifest ideas of how the mobile Internet should be utilized and how existing deficiencies might be eliminated. The Mobile 2.0 concept defends the statement that the introduction of the *Wireless Application Protocol (WAP)* was a failure, because the hypertext paradigm does not seem to be suitable for a mobile platform without the support of pointer devices. Furthermore, it is mentioned that mobile applications should be based on *Mobile Widgets* using HTTP or AJAX to communicate, as well as for storing local data and for messaging purposes. Compared to WAP clients, *Mobile Widgets* are platform-independent and customizable mini-applications with a higher responsiveness. These applications access web content more precisely, hence web browsing becomes obsolete. The *WidSets Service*[11] for instance supports this paradigm. In this sense the smart phone as a communication/social device, reacts if content is pushed to it or the user pulls content on the device. The *Mobile 2.0 Manifesto* also stands in clear contrast to proprietary technologies like *ActiveSync*, *BlackBerry*, *iPhone*, etc., because only open standards would create the prerequisites for an innovative evolution of mobile platforms and -networks. The following list gives a summary of the *Mobile 2.0 Manifesto* as published by Fabrizio Capobianco[12] [Mob06].

[9] *Wireless-Fidelity*; IEEE 802.11a/b/g networks with a data rate of up to 54Mbit/s; http://www.wi-fi.org

[10] *Worldwide Interoperability for Microwave Access*; IEEE 802.11n network with a maximal data rate per channel of 40Mbit/s and a cell radius of three to ten kilometers; planned as replacement for cable and DSL connections; http://www.wimaxforum.org

[11] https://www.widsets.com

[12] CEO of *Funambol* - http://www.funambol.com/

1. *Mobile 2.0 is NOT a mobile Version of Web 2.0*
2. *Mobile 2.0 is all about Open Standards and Open Platforms*
3. *Mobile 2.0 is driven by Open Source*
4. *Mobile 2.0 happens with Flat Fee Billing*
5. *Mobile 2.0 is centered on Content and Messaging*

Even if mobile web browsing seems to be unhandy on mobile platforms nowadays, smart phone manufacturers try to overcome this lack through advanced web interaction strategies. According to [KO08], the access rate to mobile web pages from Apple's *iPhone* platform is higher as from all other smart phones together. This is an interesting fact, since this proprietary solution has only a market share of two percent today. The adaptions of the statements like mentioned by Fabrizio Capobianco and a rapid penetration of *Wireless Broadband Access (WBA)* technologies such as 3G/UMTS as well as a blurred *All IP* mobile- and wired Internet in the next decade, would support an efficient utilization of most diverse services within the mobile IT. Moreover, new web interaction concepts and the utilization of *Web 2.0* technologies, as well as flat fee billing for end-users, might give the web a new chance in the mobile domain.

2.6. Service-Oriented Communication

Platform extensions and web-based communication models within the mobile IT domain support a flexible software development process and an integration with remote service endpoints as part of a cumulative over-all system. Based on sophisticated hardware connectivity facilities and software middleware layers, abstract service components enable a high-level focus on the actual business logic architecture of applications. Furthermore, service-oriented computing in the mobile domain is a novel paradigm in terms of allowing agile terminals to flexible utilize document-based payload interaction strategies. Service endpoints are easily exchangeable or extendable and they offer highly interoperable interface definitions for a varity of clients.

Capabilities on mobile platforms are steady growing. Built-in XML parsers and WS-I compliant (see appendix *Web Service Interoperability (WS-I)* [A.5]) protocol implementations as well as an awareness of identity-based interaction mechanisms enable a reliable integration of powerful enterprise services into the application's logic. In a similiar way, native functionalities and optional tools for accessing operator- or device-centric services are available for the application development process. Depending on the chosen programming language, those features allow a more or less fine-grained access to service parameters. Operators typically have great opportunities for offering services to their customers, since they can fall back on the functionalities of many infrastructure facilities and those services integrated by business partners. The mobile service infrastructure is however widely open nowadays, and allows the development of customized mobile solutions even without any network provider intervention. The mobile operator, third party companies, or devices within a *Personal Area Network* (PAN) may likewise provide a distinct piece of logic or some useful data for mobile platforms, within a single application context. Furthermore, operators are currently re-structuring network facilities and they start to offer web-based services as part of their infrastructure.

3. Analysis of Related Technologies

Homer: "Welcome to the Internet, my friend, how can I help you?"
Customer: "I'm interested in upgrading my 28.8 kilobound Internet connection to a 1.5 megabit fibre optic T1 line. Will you be able to provide an IP router that's compatible with my Token Ring / Ethernet LAN configuration?"
Homer: (pause) "Can I have some money now?"

— The Simpsons

3.1. Mobile Platforms

Technical issues of *Mobile Java* and the widely distributed *Symbian OS* with the according user interface platforms are an important knowledge basis for mobile Java application developers. Furthermore, this section covers *Security* aspects intrinsic to mobile platforms and applications as well as *Alternative Mobile Solutions*. The last section discusses the meaning and the usage of *Mobile Development Frameworks* for developers.

3.1.1. Java Micro Edition Platform

The *Java* programming language arose from a research project called *Green Project* in 1991. The aim of this project was to create a central control unit for electronic household devices while supporting a new developed programming language called *Oak* [Weß06]. The language was not successful on small devices, but its platform portability and its rather simply to use and quite powerful infrastructure led to a further development under the scope of *Sun Microsystems* to reinsert it to manage more advanced tasks on capable host and server machines. Next to a wide-spreaded and successful usage of Java within the *Enterprise*, Java for resource-constraint devices known as *Java Microedition* (JavaME) is nowadays also important and highly available for developing consumer- as well as business applications across multiple domains.

3.1.1.1. Architecture

Java's scalability and the platform-independent language standardization also qualifies an enterprise Java developer to create code for mobile end devices, as long as the developer is familiar with the peculiarities of the *Connected Device Configuration* (CDC) or the *Connected Limited Device Configuration* (CLDC), respectively. This is possible since JavaME is a subset of

JavaEE. The ECs[1] with are responsible for the evolution of Java Technology are divided into the JavaSE/EE and JavaME group, whereby effort is going on to build a tight integration between JavaEE, JavaME, and *Web Services*. This allows enterprise developers to build consistent end-to-end solutions by accessing intranets, databases and corporative services more easily and securely using mobile Java applications. Figure 3.1 shows the Java Platform Architecture [3GA05] with a focus on the JavaME components, which are discussed in more detail below.

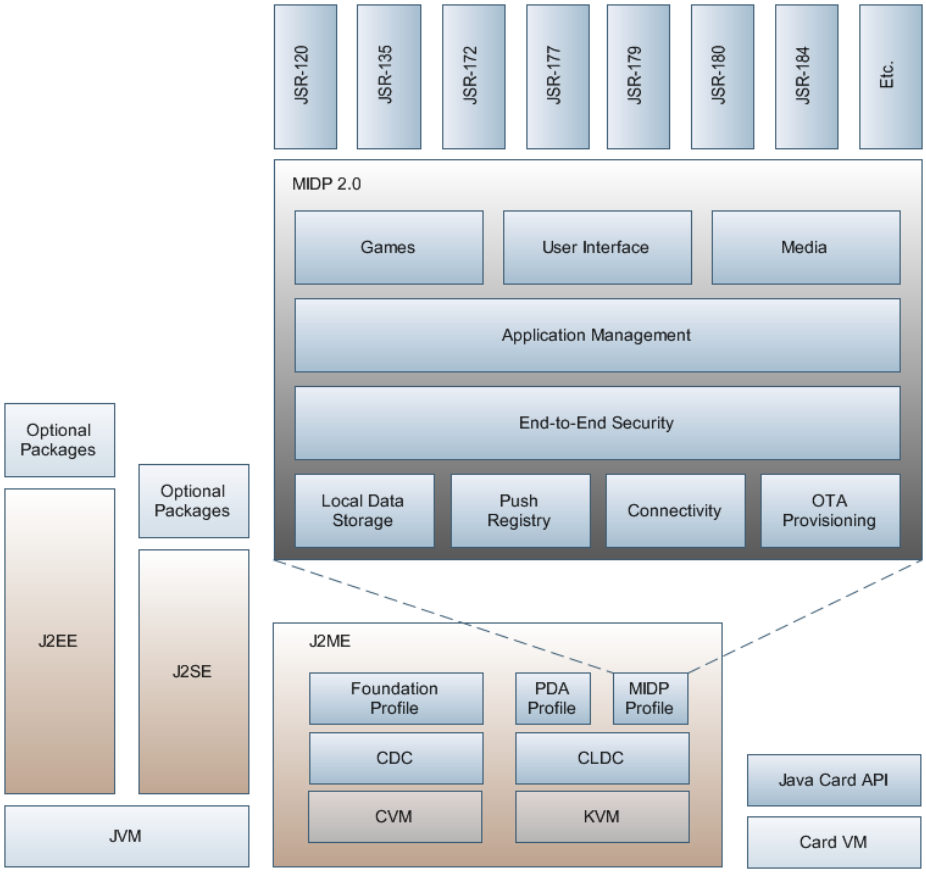


Figure 3.1.: *JavaME Platform Architecture*

As illustrated in figure 3.1, JavaME is a set of specifications and technologies, which are necessary to meet the requirements of the fragmented mobile device market. The Micro Edition architecture is based on configurations, consisting of an adapted *Java Virtual Machine* (JVM) and a minimal subset of *Java Standard Edition* (JavaSE) class libraries. The attention in this master's thesis lies on CLDC, which is the Java Platform for devices like smart phones. CLDC is running the *Kilobyte Virtual Machine* (KVM) or optimized VM versions. Profiles are used to extend configurations to provide the developer with additional libraries and corresponding *Application Programming Interfaces* (APIs) for the management of essential device features like the user interface, network connections and the lifecycle for applications. The *Mobile Information Device Profile* (MIDP) in the second version is usually used in modern devices, which is an important convergence in the industry to bring fragmented Java implementations together. The CLDC and MIDP combination provides a complete *Java Runtime Envirionment* (JRE) for executing mobile Java applications (*MIDlets*) on resource-limited devices [BM06].

[1] *Executive Committees*. An elected group of JCP members, which represents a cross-section of the Java Community. The aim of this union is to select JSRs for development, supplying RIs and *Technology Compartibility Kits* (TCKs) [3GA05].

Next to the CLDC/MIDP-based JavaME core functionality, device manufacturers and operators have the possibility to extend the profiles with optional functionality, implemented in corresponding devices. This architecture allows the best adaption of the device capabilities and constraints to a Java Environment [3GA05]. The specification of interfaces for *Optional Packages* is coordinated by the *Java Community Process* (JCP)[2] to ensure a vendor-neutral application development. Dealing with such packages decreases of course the application portability [Sch04].

The *CLDC Reference Implemenation* (CLDC-RI) and *MIDP Reference Imlementation* (MIDP-RI) are used by manufacturers as support for setting up the corresponding functionality in consonance with the underlying platform while offering the standardized Java APIs to the MIDlets. Sun Microsystems offers application developers the *Wireless Toolkit* (WTK), a much more convenient alternative to emulate JavaME device libraries from within *Integrated Developing Environments* (IDEs). It also includes frequently used optional packages, building tools, and generic visual device emulators by default.

3.1.1.2. Mobile Java Interoperability

In the year 2002, the JCP started to work on a *Java Specification Request* (JSR)[3] called *Java Technology for the Wireless Industry* (JTWI). The aim of this charter was to provide a recommendation for manufacturers, for the inclusion of specific optional packages, to create more consistent end-to-end solutions. Therefore JTWI is a seal of approval for the access of devices functionalities with Java. The JTWI is getting adapted according to the mobile device evolution. The inital JTWI specification included the *Wireless Messaging API* (WMA) and the *Mobile Media API* (MMAPI) as additional fundamental packages. A JTWI-compatible device implements the JTWI functionality and has to pass an extensive test suite [Sch04]. In 2006 a JSR called *Mobile Service Architecture* (MSA) was released to follow the same philosophy like JTWI but with a target on high volume wireless handsets [BM06]. Furthermore, each JSR provides a *Technology Compartibility Kit* (TCK) by default, which is a set of testing tools for the implementation of the specification, provided by Sun Microsystems [3GA05].

3.1.1.3. Provisioning

For the provision of mobile applications an air interface called *Over-the-Air* (OTA) was defined, to offer the customer a convenient way to obtain and install applications on smart phones by requesting corresponding web addresses within the built-in mini browser. The underlying infrastructure consists of a web based *JAD Server* holding the application descriptor files and a *JAR Server* for the corresponding application packages. The application installer on the smart phones is acting as the client. Additionally a *Notification Server* is used for getting feedback from the installer and for application updating and billing purposes. The descriptor offers

[2] Founded in the year 1998 to involve system- and telecom companies into a standardization process of Java technologies, while remaining open to improvements and innovations [3GA05]. The JCP is coordinating the expert groups, which are working on *JSRs* submitted by JCP members.

[3] Document which is managed by Sun's *Program Management Office* (PMO) for the JCP [JCP07]. JSRs may be used to initiate a new Java specification process based on expert groups. A finished interface specification usually results in a *Reference Implementation* (RI) to provide the correponding functionality and to validate the correctness of a specification. JavaME-based JSR reference implementations are usually interesting for device manufacturers. Based on JSRs, new revisions of whole Java platforms as well as small specifications are handled.

the installer relevant application properties for the installation process. The `MIDlet-Jar-URL` attribute for instance holds the web address of the JAR package. The data transfer is entirly based on HTTP [Sch04]. For more information on wireless networking issues see section *Wireless Networking* [3.3].

3.1.2. Symbian OS Platform

To satisfy mobile user demands and to efficiently integrate new system functionality, an open operating system was needed upon which developers can build advanced applications and services. In 1998, major vendors like Nokia, Ericsson and Motorola found a shared new starting point for this requirement in the *Operating System* (OS) *EPOC*, which envolved to *Symbian OS* later on. Nowadays, Symbian OS has the largest market share and is the industry's standard choice for smart phones, since it also supports enterprise information systems.

3.1.2.1. UI Platforms

Unlike company-centric proprietary systems, Symbian OS is usually more rich in functionality by making use of advanced *C++* features, *Object Oriented Programming* (OOP) and APIs [Sym07b]. Symbian offers several *Software Development Kits* (SDKs) for building C++ and Java applications, including binaries and tools like the *Universal Emulator Interface* (UEI) [dJ04c] to integrate with Sun's WTK and IDEs for emulating MIDlets as well as deploying capabilities, APIs and system documentations. SDKs are based on *UI Platforms* associated with device *User Interfaces* (UIs) and a set of system applications, which are typically making use of generic application engines of the OS. Installable, third-party applications (`SIS` and `JAR/JAD` packages) have to be maintained by an SDK of a particular UI Platform, which is related to a distinct OS version and supported by device manufacturers and by devices with according capabilities, respectively. Common Symbian OS UI Platforms are *UIQ*, *Nokia S60*, *Nokia Series 80*, and *Nokia Series 90*. S60 has the biggest market share and is target for voice-centric smart phones with information capability, supporting C++, Java, and *Flash Lite* by default. UIQ and Nokia Series 80 based phones are information-centric and are therefore more suitable for enterprise services. Series 90 phones introduce more advanced interaction features, like touch screens [Sym03]. To ensure maximum interoperability, all these platforms have a Symbian OS specific shared published API set, which is however often extended by device vendors to address their special needs for the phone's UI charateristics [Sym07a].

3.1.2.2. Development Environments

Several free tools are available for a standalone usage as well as for IDEs to support the programmer and designer with a convenient graphical interface for accessing specific SDK functionalities. Nokia for instance provides the so called *Carbide* development tools for this purpose. *Carbide.c++* is a bundle of *Eclipse*-based tools for Symbian OS development on S60. *Carbide.vs* supports Microsoft *Visual Studio* in the same way and *Carbide.ui* is a full WYSIWYG[4] layout tool for mobile application designers. Nokia also provided a bundle for Java development called *Carbide.j*, but since the popular combinations of Eclipse IDE with *Eclipse*

[4] What You See Is What You Get

ME and Sun's *NetBeans IDE* with *NetBeans Mobility Pack* are suitable to offer similar features, Nokia's Java solution has become nearly obsolete [Nok07c]. Nokia also offers the so called *Prototype SDK for JavaME* as optional package, which does not emulate a full JavaME platform functionality, but instead offers a lightwight SDK and code examples specific to Nokia's Java reference implementation. This SDK contains several UI platforms emulators and provides the developer with an earlier access to new specified Java APIs and a higher performance compared to Sun's WTK [Nok07c]. However this APIs may have been changed in SDKs later on, whereas final MIDlets should be based on officialy released Java SDKs corresponding to real device implementations. Symbian OS also supplies exclusive API extensions for C++ and Java to access system functions not currently available in SDKs.

3.1.2.3. Introduction of Version 9.0

In 2005, *Symbian OS Version 9* was introduced, whereby *S60 Third Edition* and *UIQ 3.0* as newly released UI platform versions aim to support an entirly new market landscape, based on a new Symbian OS concept. Due to a radical architecture change based on new approaches like a scalable UI architecture, a re-writing of device drivers was necessary which is why the S60 Third Edtion is *not* binary compatible with S60 Second Edition platforms. The *S60 Third Edition Feature Pack 2* package furthermore introduces concepts like *Web Runtime* to allow access to *Web 2.0* features on smart phones [Nok07c].

3.1.2.4. Native Security Concepts

Essential changes at the design, architecture, core, and management level are coming along with *Symbian OS Version 9*. Next to general improvements, a support for OMA standards (see appendix *Open Mobile Alliance (OMA)* [A.1]), a new kernel for supporting realtime applications, and an enhanced *Symbian OS Security Model* should better meet the requirements while retaining the benefits of an open platform. The security framework delivers restricted access to sensitive APIs and therefore to underlying trusted OS applications by defining *Levels of Trust*. About 40 percent of the OS API access are managed by this so called *Trusted Computer Base* (TCB). These APIs are therefore signed and mapped to so called *Capabilities* (see appendix *Platform Security Capabilities* [B.1]), which represent grouped types of similar protected functionality. A direct access to Capabilities without user intervention is only possible for trustworthy applications. In all other cases the user will be confrontated with a security warning and the application has the most restricted access to the system functions. This strategy makes sense to protect the user's privacy, the device, and the network, but may be annoying while developing and testing applications accessing signed APIs. For *S60 Third Edition* signing is mandatory in order to be able to install *native* application on the platform, even if no protected capabilities are used. The corresponding S60 Third Edition SDK provides therefore a self signing function to get installable applications and to grant them access to essential resources. The propably best solution for a developer of *native* code using capabilities is to order a free *Developer Certificate* (DevCert) offerd by Symbian's official testing program. This certificate allows individual Symbian developers for a period of six months a trusted access to protected APIs on a single device, identified by its unique *International Mobile Equipment Identity* (IMEI) number [Car].

3.1.2.5. Java Security Concepts

MIDP 2.0 defines a specific *JavaME Security Model* with four possible protection domains for MIDlets, whereas signing is not mandatory. Applications belonging to the *Untrusted Third-Party Domain* are not signed and display a warning before the installation starts as well as permission requests before accessing capabilities. MIDP 1.0 applications are untrusted by default. In the *Trusted Third-Party Domain* applications are official tested and signed. However, this domain does not remove all permission intervention with the user completely by default. In this domain the permission dialog behaviour may be influenced by the user by configuring the appropriate system settings, whereby all permission dialogs can be disabled for trusted ones and a some few ones for untrusted domain applications. *Operation Domain* applications have free access to all underlying APIs without asking for any user permissions. Finally, with *Manufacturer Domain* applications also accessing hidden APIs is possible. In the last case the manufacturer is responsible for signing such specific applications [Nok05]. Signed MIDlets are allowed to request permission for sensitive APIs quoted by the developer. The two attributes `MIDlet-Permissions` and `MIDlet-Permissions-opt` are therefore available in the application's *Manifest* and in the according descriptor. Using the `JAD` approach for defining required capabilities is usually making more sense, because in this case it is possible to check the existence of the necessary platform functionality already during the installation phase. If this check fails or the user declines a capability access, the installation will be cancelled and the corresponding `JAR` package is not downloaded [Sch04]. Moreover, the nature of Java application execution as intepreted byte code within a *Sandbox* (cf. JVM) protects the system additionally against malicious code.

3.1.2.6. Deployment

For *Production* purposes, the final application must be passed through the official corresponding Testing Program in order to ensure an adequate quality of the end product and to get digitally certificated. The OS *Application Management System* (AMS) installer performs a verification of this *Digital Signature* (key) against a device intern *Root Certificate* (lock) from a trusted *Certificate Authority* (CA). A correct signature allows applications accessing protected functionality on compatible Symbian OS devices as long as the certificate is valid (usually one year). Symbian also offers developers to sign their final applications as freeware without any costs as long as the program is innovative enough and providing its content in the `SIS` or `PKG` format. This also includes non C++ projects as long as the according development tools support this package format [Nok07a]. In 2004 *Symbian Signed* as official testing program for native applications and *Java Verified* as official testing program for MIDlets have been founded to prevent a further fragmentation of the existing testing area at this time [Nok05]. These two programs are the industry-wide agreed standard testing programs which allow mobile applications to be distributed on mass-market devices.

The *Symbian Signed* testing and certification procedure presumes three steps. First the publishing developer or organization a.k.a the *Independent Service Vendor* (ISV) obtains an *Authenticated Content Signing* (ACS) Publisher ID from *TCTrustCenter* which is the Certificate Authority chosen by Symbian [Sym07a]. This Publisher ID effectively provides assurance of organisational identity. Second, the publisher signs the application with this ID certificate and pre-tests against the *Symbian Signed Test Criteria* on own devices. As a last step the application is submitted to an registered and independent test house for a testing phase of around one week to get the

certificate. Symbian's trusted partners and large ISVs have also the opportunity to join the so called *Shelf Certification Program* as a replacement of the last required step [Car].

For MIDlets the *Java Verified Program* (JVP) ensures an adequate level of code quality through the *Unified Testing Initiative* (UTI) guidelines. A MIDlet is signed with the *UTI Root Certificate* if the candidate fulfills the UTI test criteria. This certificate is also stored on the corresponding device for validation purposes later on. The MIDP specification does not allow to add new certificates on S60 Third Edition platforms [Jav07]. In the signing procedure, first the submitted `JAD/JAR` file content is examined and pre-tested by the ISV. As a further step, the ISV's chosen testing house uses the asymmetric cryptograhic *RSA* algorithm to sign the application to provide assurance of organisational identity. Therefore, a *X.509 Public Key Infrastructure* (PKI) certificate [Jav04], related to the ISV and signed by another trusted CA (e.g. Verisign), is used. In the actual encryption step the application's checksum is signed by the ISV's private key and stored in the `MIDlet-Jar-RSA-SHA1` property field of the descriptor file as a *Base64* encoded string value. This unique value is the signature of the ISV. Under normal conditions this value is sufficient to prove the authenticity and the integrity of the application by decoding and decrypting it with the ISV's public key. Since this requires the AMS's awareness of every public key of a possible ISV resulting in a large list, another approach is necessary for resource limited devices. For this reason, the MIDP 2.0 application descriptor also defines a `MIDlet-Certificate` property field for storing the *UTI Root Certificate*. In MIDP 2.0 it is even possible to store certificate chains in the descriptor. Finally the ISV gets back the signed and certificated application through the JVP program. Since the application signature is based on the content's checksum, a MIDP 2.0 compliant device can also check its integrity to detect unauthorized code manipulations. This also means that updating the content (including non-code resources) by the ISV will invalidate the signature [Sch04]. Figure 3.2 illustrates the verify process and the tasks of the involved parties.

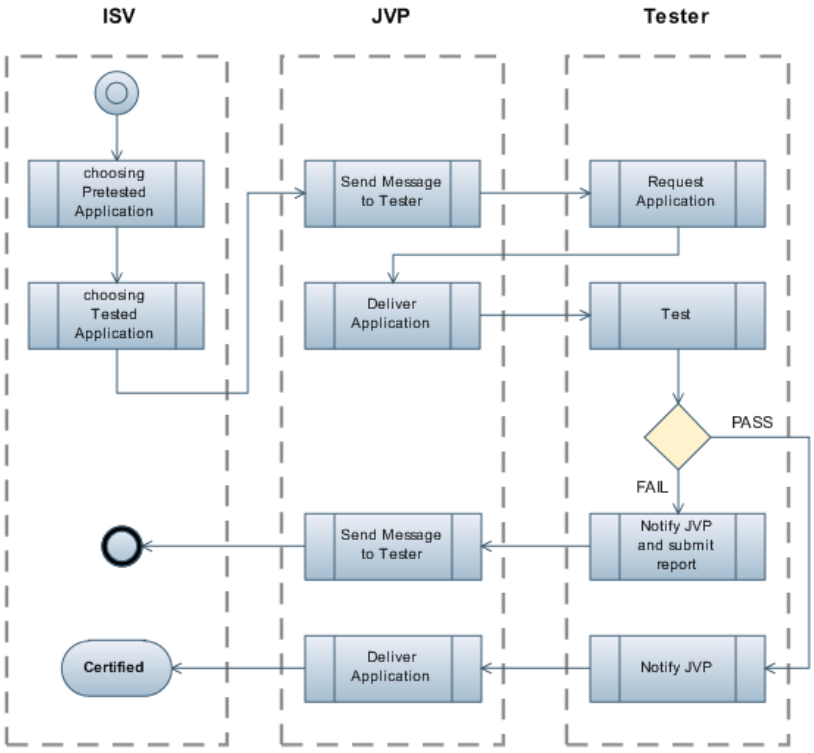


Figure 3.2.: *Workflow of signing MIDlets with the Java Verified Program*

3.1.2.7. JVM Support

As mentioned in section *Mobile Java Interoperability* [3.1.1.2], every new JSR specification must pass a TCK. However this does not prevent different implementations of the same JSR. Often vendors provide the mendatory JSR functionality in subtle ways and optional JSR features are implemented arbitrarily. Due to this fact, it is sometimes important that Java developers know implementation-specific gaps even when creating standard JavaME applications for Symbian OS platforms.

Java development protects the programmer from dealing directly with the complexity of OS specific features, but the use of high-level Java APIs is sometimes a restriction in case of reqirements for a specific OS functionality to solve a distinct problem. On the other side, more expensive C++ solutions have full feature access capabilities while decreasing execution latency, since they run as native OS processes [dJ04a]. Anyway, Java's portability and security model are crucial aspects for the wide support of Java Virtual Machines (JVMs) over many platforms and the precense of a huge Java application development community.

Symbian OS is a fully multitasking system providing multiple processes and included threads. The OS is based on a *Client/Server Framework* whereby the server offers services to corresponding clients. Active objects are scheduled by a single *Event Handling Thread* instead of spawning threads on requests, like in other operating systems. Symbian OS actually supports a port of Sun's *CLDC HotSpot* implementation of the VM (refered to as *CLDC HI VM*) for improving performance of MIDlets, compared to the former KVM implementations. The CLDC 1.1 HI VM is also able to handle floating points, which offer a wider application spectrum. The VM itself runs as a native thread within its own process. To enable Java components to receive callbacks from native code a so called *Java Event Server* runs externally to the VM as a native thread, which makes use of the OS Client-Server infrastructure. Java threads are either mapped to underlying OS threads or represent platform-independent lightweight threads of the VM. An OS supporting CLDC HI VM, only uses the portable lightweight thread model. Unlike the CLDC HI VM, the MIDP environment is implemented by Symbian, whereby an optimal performance for Java applications and a similar behaviour and *Look-and-Feel* like their native counterparts can by ensured. Consequently Symbian's CLDC/MIDP implementation is based on JavaSE and the *Java Native Interface* (JNI) framework. Furthermore Symbian only imposes restriction for MIDlets in terms of memory limits. The amount of *Record Management System* (RMS) records, Java threads, and socket connections are arbitrarily defined by the developer. There is also no limit of the installable MIDlet `JAR` file size and the initial CLDC HI VM heap size of usually 400kB is adapted dynamically while executing MIDlets [Sym].

The OS *Application Management System* (AMS) controls the creation, starting, pausing, and deconstruction of MIDlets and switches therefore (between) the MIDP-specified lifecycle states `ACTIVE`, `PAUSED`, and `DESTROYED`. As soon as an executing program has to be interrupted, due to allocating device resources to higher priority tasks, the `PAUSE` state ensures that the automatically backgrounded MIDlet only consumes essential resources and releases shared ones [dJ04b]. However, in practice this behaviour is not always controlled by the AMS logic, since the circumstances under which the corresponding `pauseApp` method is called, vary among UI platforms designs. For UIQ the AMS is acting like expected, but not so the AMS for S60 platforms. In this case the developer has to provide the necessary functionality by monitoring the MIDlet's display state, to activate the *PAUSED* state manually [Sym]. In the `DESTROYED` state transient MIDlet data should be eligible for garbage collection. [dJ04b]

Application runtime errors and exceptions may originate from Java or native code. If abnormal conditions are encountered in native code, function calls provide an integer error code (C++ exceptions are not supported yet) to the calling Java method. In succession a Java exception appended with the *Symbian OS Error Code* is thrown. Since `System.out` and `System.err` are absent in JavaME implementations, on-target debugging is unconvenient by default. Moreover, it is crucial to deploy and test the MIDlet frequently and in an early development stage, since software emulators are often unreliable and do not mirror the exact device behaviour [Sym].

In 2007, Sun introduced the *JavaFX* product family. This technology is targeting content developers and application designers by enabling an easy and flexible enrichment of applications with multimedia content and advanced user interfaces. JavaFX and *JavaFX Mobile* provide features known from Swing, Java2D, Java3D, JMF, etc. within a script language, but with optional islands of pure Java code. JavaFX might become a serious technology in the mobile domain competing Adobe's *Flash Lite* and Microsoft's *Silverlight* implementations.

3.1.3. Alternative Platforms

Additionally and concurrently to the Symbian OS, several other open mobile platforms like *BREW*, Linux-based Systems, *Windows Mobile*, *Palm OS*, etc. as well as the proprietary approaches of *BlackBerry* and Apple's *iPhone* are under steady development and usage. Some of them are also targeting at the non-smart phone domain by providing mobile frameworks for building software systems on devices with more or specialized capabilities like Internet tablets. Figure 3.3 gives an overview of mobile platforms and commonly available development environments.

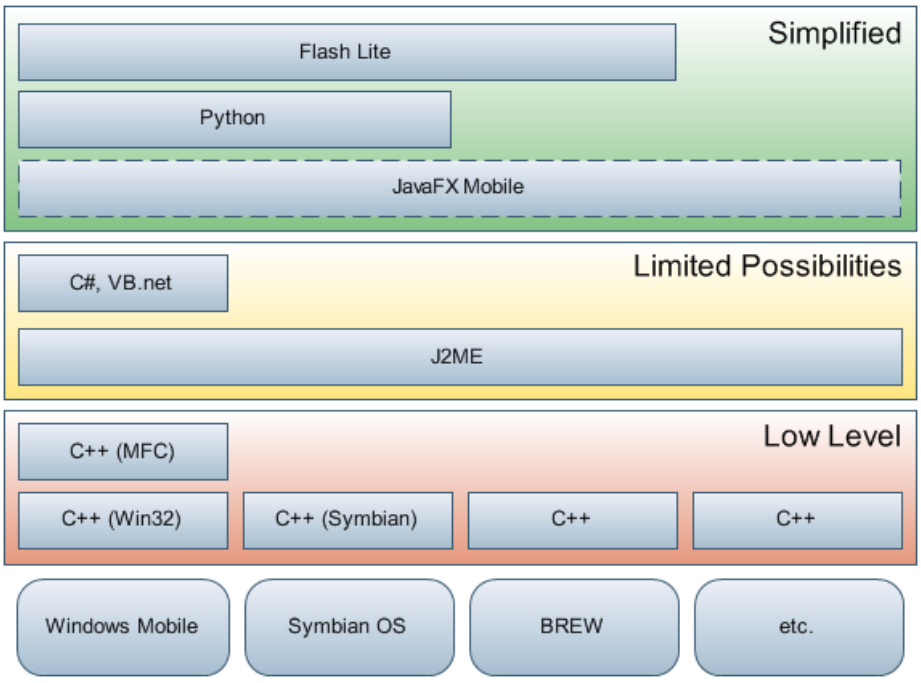


Figure 3.3.: *Mobile Platforms and their Association with SDKs*

3.1.3.1. Symbian Extensions

Since Symbian OS is a non-proprietary operating system it is scalable for supporting environments for language independent code execution. Runtime environments, which are ported to the mobile domain often arose from open source projects based on languages like *Python*, *Ruby*, *Perl*, *OPL*, *Simkin*, etc. [Sym07a]. *Python for S60* a.k.a *PyS60* was initiated by Nokia and is frequently used for rapid prototyping purposes and also supports on-device programming. Phyton scripts can be deployed as libraries or as executable `.sis` applications, through the usage of conversation tools [Mob07]. In 2007, Symbian introduced *P.I.P.S Is POSIX on Symbian OS (P.I.P.S)* libraries [Sym07c] for supporting the migration of existing commercial and open source desktop and server components (E.g. middleware, web servers, and database systems) as well as mobile applications to Symbian OS. P.I.P.S therefore enables easy *Open C* application porting and developing, while bringing the benefits of Linux to the S60 platform. About 75% of the Open C APIs, which are based on well-established open source projects within thousands of desktop-, server-, and embedded software products, are available on the mobile platform. Open C development also avoids a time-consuming incorporation into Symbian specific C++ APIs. P.I.P.S and the Open C extension are now available as plug-ins for third edition devices and will be firmely integrated into Symbian 9.5 smart phones.

3.1.3.2. Linux

Founded in 2005, the Nokia hosted *Maemo*[5] platform is an open source project used to mobilize *GNU Linux* based desktop applications using Nokia's Internet tablet devices. The aim is to have a full Internet experience in pocket-size format by supporting *Wi-Fi* Internet connections and a rich user interface in form of the *GNOME* based *Hildon UI*, developed for resource limited devices. Furthermore a full *Mozilla Gecko Engine* provides a sufficient web runtime on the tablets. Within the mobile Linux world, *BlueCat*[6] by *LynuxWorks* is another approach to mobilize applications with a supported realtime performance on smart phones, by using the *kernel 2.6* as underlying engine.

As a further promising Linux-based platform in an early development stage right now, *Android* represents an open software stack for developing mobile Java applications. The Android platform was initited by *Google* as an open handset alliance with 33 partners from the IT industry and Google expects an availability on the market under a *Apache v2 Licence* in late 2008. Based on the 2.6 kernel Android will include libraries from other open-source projects and the *WebKit* web engine which is already used for web browsing on Symbian S60 and Apple's iPhone. As a virtual machine, Android introduces *Dalvik VM* which is a register-based VM optimized for resource-limited devices and proprietary code execution, not suitable for interpreting standard Java byte code. It is therefore not compatible with JavaSE and JavaME[7]. Application development is primary based on Google's *Android* packages including the graphic libraries. All applications for Android are developed in the Java programming language, regardless of whether these are system components or third-party consumer applications. According to Google, this strategy will enable the development of mobile *Mashups* for a communication between different applications. The development of an own device for the Android platform following the iPhone concept is expected [Gla08].

[5] http://maemo.org
[6] http://www.lynuxworks.com/embedded-linux/embedded-linux.php
[7] Only the `java.nio`, `java.util`, and `java.lang` packages will be available for development.

3.1.3.3. BREW

The *Binary Runtime Environment for Wireless* (BREW) by *Qualcomm* is an open platform approach with many possibilities for a wide-spreaded spectrum of mobile IT players. The BREW C++ based framework has low level hardware access capabilities by supporting a language neutral application execution including MIDlets. With this *end-to-end* solution it is even possible to install JVMs on demand, which are handled as ordinary applications by the BREW infrastructure. Due to the *BREW Distribution System* (BDS) as a key element with wireless data service delivery and billing features, Qualcomm created a suitable business model for the wireless economy. After passing a BREW specific test, applications can be browsed and OTA downloaded from the operator's application repository. The platform uses a virtual handshake between the handset and the BREW *Application Download Server* (ADS) to meet installation requirements including a transparent JVM extension download for running Java applications on non-VM resident BREW devices. BREW therefore offers a flexible virtual marketplace for developers, operators and customers. Qualcomm argues that BREW provides a seamless layered device interaction from high level to low level platform functionality, in contrast to stand-alone operating systems with additionally third party platform components, which should allow BREW applications a much easier device portability [Qua03]. The BREW approach has indeed several advantages, but nevertheless its development is bound to a single company, which naturally excludes rapid innovative implementations necessary for a dynamic mobile market. Since a JVM is natively shipped with nearly every handset nowadays, deploying Java applications on those devices is not really a handicap, which makes the descript BREW concept almost obsolete by even additionally introducing more overhead when executing Java applications on BREW platforms.

3.1.3.4. BlackBerry

A further innovative *end-to-end* mobile solution with a focus on secure mobile enterprise and standard Internet services offers *BlackBerry*[8], which is a device vendor as well as a platform and service provider. BlackBerry's research subsection *Research in Motion* (RIM) provides the necessary hardware components. BlackBerry implementations cover client- and server-side components for seamlessly accessing email accounts, enterprise data, web pages as well as enabling standard voice and messaging services. Even if the company manufactures own smart phone series, the architecture of the *BlackBerry Enterprise Solution* for companies and the *BlackBerry Internet Solution* for medium wireless business solutions and individual services also enable access to the BlackBerry infrastructure from foreign operation environments by using the *BlackBerry Connect* platform extension. Within the web service based enterprise domain a specific *Enterprise Server* (BES) for involving *Mobile Data System* (MDS) applications is required. Since standard network protocols are incorporated, existing server-side applications can be extended to operate with thin- or thick client MDS' regardless of the underlying technology by providing a centralized administrative console for managing system aspects. The BlackBerry smart phone clients run a Java-based operating environment with a customized BlackBerry JVM for only supporting a MDS-specific MIDlet execution environment. The development of enterprise and MDS applications is supported by the BlackBerry MDS SDK as well as through a corresponding IDE support (*BlackBerry MDS Studio*), Microsoft Visual Studio extensions and tools.

[8] http://na.blackberry.com/eng/

This wireless proprietary *all-in-one* solution offers a consistent and scalable mobile enterprise, Internet and electronic mail concept, but since it is a closed and company-coupled approach the corresponding company business models are close tied to a specific technology (e.g. only distinct mail servers are supported). Furthermore MDS application developers have restricted capabilities of building a customized infrastructure according to potential future needs or an additional money investment in foreign software components and licences will be necessary. Nevertheless BlackBerry has an advanced security model which is according to [Bla06] usually more suitable for enterprise information systems than the security concepts of ordinary smart phones. The 3DES or AES encrypted convidential data transfer from a static BES server to a mobile BlackBerry client is routed over three worldwide distributed and RIM controlled *Network Operation Centers* (NOC) to the distinct mobile networks. The necessary keys are protectively stored only within the communication end points, unaccessable through third party MIDlets [A-S04]. Several case studies performed by *Symantec, Secorvo and CESG* attested the BlackBerry security concept as sufficient. In 2006, BlackBerry's security arrangements were even certificated as government and NATO suitable [Bla06].

3.2. Mobile Software Development

In this section the peculiarities of developing *MIDlets* for the wide-spread *Symbian OS Platform* will be discussed and the pilosophy behind corresponding frameworks and tool suites will be presented in order to maintain a suitable and efficient development process by supporting advanced mobile concepts, reusability and reducing the time to market. Therefore an important question for ISVs usually related to mobile application development will be:

> *'If we write an application for one Symbian OS phone, how much work will be involved in getting it running on a different Symbian OS phone, running possibly on a different UI platform?'* [dJT04]

3.2.1. Code Efficiency

The heterogenity of mobile platforms with different vendor-specific device features is one of the main reasons for the introduction of independent mobile frameworks and tools to overcome some of these problems. Furthermore the MIDP specification leaves not much space for an adaption of the application development process and a customization of the mobile application in respect to extensions. Some known limitations for the developer are the poor support for a reliable debugging, an unchangeable standard *Look-and-Feel* for LCDUI software components, and the unmanageable varity of aspects to consider while developing cross-platform applications. However, the usage of extending frameworks during the development process normally introduces some overhead in memory and CPU cycles, whereby the developer should be aware of methods to reach the best ratio between efficient code execution and the maintenance of the desired functionality. Particulary a solid knowledge about the underlying executing environment, about its properties and restrictions is always an important aspect for a software developer when designing and writing code, and still much more important for a mobile and embedded software programmer.

3.2.1.1. Application Design

Independent of a particular extending framework, a *native* MIDlet suite should ideally be designed in a way, to maximize its performance and to minimize its memory footprint as well as to meet the requirements of mass market, cross-industry devices. Due to this fact, deploying mobile applications is based on development and optimization of code for different platforms by default. In this respect, it is also important to consider that a code target for possibly the S60 platform is hard to modify to run on less advanced Series 40 devices. This is often a crucial and necessary aspect especially when developing *Business-2-Consumer* (B2C) applications related to a huge user base and several UI platforms. Faulty implementations in firmwares can furthermore lead to unpredictable failures when accessing specific device features and sometimes even carriers modify devices to support provider-specific services. Therefore, designing the implementation process is a challenging task, where to find some common ground on which to base the central application features. This functionality is then optionally extended through more specific JSR-APIs and frameworks, optimized for a subset of platforms or even for common device models. This results either in one application bundle per device which is uneconomic in terms of application maintainance and signing, or one `JAD/JAR` bundle for all devices [Vir05]. In the second approach, the device variances are handled by the application logic at runtime. The described strategy is supported by the JTWI and MSA initatives mentioned in section *Mobile Java Interoperability* [3.1.1.2].

3.2.1.2. Execution Speed

The code execution speed of MIDlets depends on several aspects and some of them can be influenced more easily than others. In general one known issue is that Java programs spend about 90 percent of their execution time in 10 percent of their code and a major part of the program deals with function calls to external libraries [Nok04a]. For that reason it is more suitable to find bottlenecks in the source code rather than to optimize each line of code. Furthermore, MIDP implementations are vendor specific, whereby their performance varies among each device and even among different versions of the same model. One way to measure performance and to find bottlenecks respectively, is to use the *WTK Profiler* tool as a part of the WTK emulator. It generates statistics during code execution in terms of method calls and CPU utilization. However, this measurement must always be seen in relation to a device platform with possibly 200MHz clock rate and a few mega bytes of heap memory. On real devices, bottlenecks may be verified by enclosing suspicious code within `System.currentTimeMilis()` statements and observing the result through some kind of logging strategy. To overcome performance problems, a balanced multi-threading strategy, a replacement of the underlying network protocol or avoiding recursive algorithms are typical points of improvements when executing code on an MIDP platform. Unfortunately, managing multiple threads also increases the application complexity and decreases code maintainability. Also `static` and `synchronized` access modifiers as well as Java's synchronized container data structures like `vector` or `stack`, should be used rarely in respect to performance and memory issues [Vir05]. To prevent a possible code reengineering later on, MIDP-specific performance aspects should already be considered during the application design process.

3.2.1.3. Network Connections

As mentioned in the section above, performance considerations for cellular network-based connections are another important issue, when designing a mobile application. Avoiding unnecessary data traffic speeds up the application, increases the overall usability and prevents additional service costs for the end user. The connection bandwidth has usually a major impact on networking speed when dealing with a huge amount of data, while networking latency is more relevant, when exchanging small data chunks between network nodes. As with other time-consuming tasks, a network interaction should run within a separated threaded environment to prevent a blocking of the main thread during the operation. Within the mobile networking domain asynchronous service connections are usually not available. Even if a *Domain Name Service (DNS)* allows a flexible remote service access, the name resolution slows down a service and a hardcoded addressing is often the better choice within MIDlets[9] [Vir05]. Due to the higher HTTP round-trip times of wireless networks compared to the wired ones, a *Proxy Servlet* or a similar approach might be used to gather all necessary information for sending the requested data in one go. Such an intermediate node may also be useful if a MIDlet is otherwise forced to handle complex XML-based network protocols directly [Nok04a]. However, nowadays rudimentary but efficient XML parsing and messaging on high-end devices can be done without a big overhead, which moreover provides the foundation for a pure and flexible integration of the mobile device in service oriented infrastructures. This offers a varity of new application opportunities (see section *Service-Oriented Computing* [3.4].

3.2.1.4. Application Size

Unlike ordinary software packages, the file size of the final mobile application package should be as small as possible to reduce OTA download costs and to prevent high memory consumption on the device. During the design phase this might be considered by composing functionality and therefore reducing the number of classes, while finding a trade-off between application size and an adequate application structure corresponding to a particular problem. For a feature-rich mobile application it is however wiser to build an intuitive, modular and well-maintainable package-based implementation structure, rather than caring too much about the size of the application itself. Due to the additional overhead, the heavy usage and implementation of OO-design patterns and abstractions should also be somewhat restricted within a resource-limited environment, as well as an unnecessary deployment of full MIDP libraries, if their functionality is not completely exploited during execution. In respect to the package size, *Obfuscators* are the most useful tools during the building process. Depending on their implementations they remove unused code and efficiently map the content of the source code to equivalent, but significantly shorter character sequences. This usually results in a faster code compilation and class files, which are about ten percent reduced in size [Nok04b]. Decompiled obfuscated byte code generates hard to understand source code, protected against unwanted reuse. Finally, application resources such as images should be as small as possible.

[9] Especially when it can be assumed that the service resides on the same machine for a longer period of time.

3.2.1.5. Memory Consumption

The memory heap size for MIDlets may reach from about 150kb for MIDP 1.0 devices up to several mega bytes for high-end MIDP 2.0 phones. Since MIDlets are able to run concurrently on S60 devices, requesting a guaranteed value for already allocated resources from the runtime is not possible. To efficiently utilize the shared and expensive memory resource and to accelerate code execution, reusing objects should be favoured over creating new ones and outdated object references should be released for garbage collection whenever possible. Therefore, it is wise to use some kind of clean-up tasks within the code, to set currently referencing variables directly to `null` during runtime. This also avoids memory leaks. The CLDC HotSpot VM garbage collection process is releasing dynamic memory quite effectively, by handling about a thousand of unreferenced objects per second [Nok04a]. Consequently, the developer has to find a good trade-off between execution speed and memory consumption.

3.2.1.6. Application Testing

Software emulators and corresponding tools shipped with Sun's WTK and UI platform development kits may support mobile application development. However, such features should be used with care, due to unmapped execution differences to real devices and erroneous implementations. In this sense, verifying application code on a varity of real devices is essential before releasing a product to the market. Nokia offers therefore the so called *Remote Device Access (RDA)* technology [Nok07b], which is a distributed testing infrastracture consisting of a Nokia device pool, situated in a lab. When using this free-of-charge service, the developer is able to deploy the application bundle directly on the remote devices via an Internet connection, whereby the corresponding device screen is transferred to the local desktop in realtime. In this way, implemented features may be efficently tested without the need to have several devices in the actual surrounding.

3.2.2. Frameworks and Tools

As mentioned in the section *Motivation - Mobile Internet* [2.5.3], it is considered that the mobile platform is currently within an evolution process towards a changed web accessability paradigm called *Mobile 2.0*. Frameworks and tools are supporting various stages within the mobile software development cycle and may be the driving force for building advanced mobile applications not only for the Mobile 2.0 era, but also for contemporary services.

Mobile frameworks address issues of *Device Management*, *Graphical User Interfaces (GUI)*, *On-Device Debugging*, *Code Maintainability*, *Service-Coupling*, etc. and are therefore extending mobile standard implementations. According to [Vir05], *'Thanks to the different behaviour of the devices, creating stable APIs for J2ME is quite a challenging and complex task'*. A developer should try to use proven and stable third-party APIs instead of creating own implementations whenever possible. However, intrinsic to the integration of any third-party products into own projects, the developer's final work also depends on the support through other vendors. In the worst case, serious bugs may be part of a framework over a long time which usually restricts constitutive code. In the next sections some promising framework implementations are briefly

described. All of them aim an efficient software development process which also considers time to market issues.

3.2.2.1. J2ME Polish

The architecture of the open source mobile development suite *J2ME Polish*[10] from *Enough*, extends the JavaME implementation and is divided into a *Build Framework*, a *Client Framework*, *IDE Plug-ins*, and *Stand-Alone Tools*.

The *Ant*-based Build Framework extends the project's Ant file and enables a customized code *Pre-Processing*, as well as an XML-based project configuration and a framework-specific resource integration. An exhausting and ajustable mobile *Device Database* in form of an XML-based document, is one of the resources which may be included in the building phase. This database allows the developer to customize a mobile project for a specific device or UI platform with the help of code-intern pre-processing variables and stored device capabilities. The *Locale* of the application's GUI and shared global variables may be defined through further documents. The build therefore results in a tailored JAD/JAR bundle, which should be portable on the aimed devices without any problems. With this strategy different versions of an application may be deployed with less effort. The framework even considers known platform-specific bugs, which is however no guarantee for a proper deployment on real devices. One of the most useful components is the *Logging Framework*, customized and also deployed by the developer on the handset as part of the application bundle. The implementation is reacting on `System.out.println()` statements annotated with pre-processing variables, and configured to be called on thrown exceptions by default.

The Client Framework consists of APIs for an advanced mobile Java development. The most important and most innovative one is an extension of the MIDP's high level API. This GUI extension separates the UI design from the application logic, which improves the code maintainability. The designer's work is only based on the definition of an according *Cascading Style Sheet (CSS)*[11], which refers to the application's UI components (e.g. screens, text fields, choice groups, alerts, etc.). This web-based design concept enables a much more customized *Look-and-Feel* than a pure MIDP approach (e.g. transparent items, animated backgrounds and menus, tabbed and framed forms, etc.) and therefore a greater user experience. The Client Framework also covers a *Game Engine* and *Utility Classes* to support for instance some useful JavaSE programming features. JavaME Polish also enables an extention of the framework itself through third-party APIs or own Java-based libraries. The framework may be seen as the next step of the open-source approach *Antenna*[12]. Antenna is also a mobile framework based on Ant tasks and delivers a similar functionality as the Build Framework from Enough.

3.2.2.2. Enterprise Integration

For distributed *Business-2-Business* (B2B) applications, aspects like data accessability, data synchronization, service security, service agility, database management, a rapid development,

[10] http://www.j2mepolish.org/; The framework's name still refers to the former notation of the Java Microedition Platform. Since the product is well known under this name, the master's thesis refers to it as such.

[11] Also allows the usage of *pre-defined*, *static*, and *dynamic* styles.

[12] http://antenna.sourceforge.net/

and a long-lasting inter-domain communication structure are typically more important than enhanced GUI features and the support of a maximum of devices. Companies and research institutes typically develop, involve or extend abstract middleware layers to customize complex business models and mobile information systems according to their needs. The resulting implementations might be divided into several adaptive and loosely coupled service components which encapsulate platform- and network specific implementation aspects like described in the sections above. Such systems are often based on proven middleware implementations, specifications and tools which usually support an efficient application development process. Representives of such platforms for the mobilized enterprise domain are for instance *Open Mobile IS*[13], *EQUIP*[14], and the *Multichannel Adaptive Information System* (MAIS) framework [Per06], which are however not discussed in more detail, since they are not of pratical relevance within the master's thesis. Instead, the thesis will present the architectural concepts behind such large-scale systems (see section *Service-Oriented Computing* [3.4]).

3.3. Wireless Networking

3.3.1. Cellular Networks

The elimination of the monopolistic telecom infrastructures and the formation of competitive network providers in the 1990s led to an international acceptance of the *Global System for Mobile Communication (GSM)* and technology-related standards. GSM and the increasing miniaturization of digital components resulted in a further price collapse for end user devices. Nowadays wireless packet-based wide-area networks and corresponding interfaces are established, which are the foundation for more advanced cross-network information exchange strategies and a variety of potential mobile services.

Due to physical and terrain-specific[15] conditions, the electro-magnetic signal strength decreases with the *fourth* potency to the distance from an emitting antenna, resulting in an additional complexity while maintaining a cellular network. Furthermore, overlaying signal transmissions cannot be isolated like in the case of wired channels. A reliable wireless communication for a huge user base is therefore only possible through expensive time- and frequency multiplexing methods[16] [Rot02]. The complexity of cellular network infrastructures typically leads to high investments for the operator.

3.3.1.1. GSM

The *Groupe Spécial Mobile* founded in 1982, defined the GSM standard and joined the *European Télecommunication Standards Institute (ETSI)* as *Technical Committee (TC)* in 1989. GSM was introduced in 1992 as mobile communication standard of the *Second Generation (2G)*. The digitalized 2G network is designed to support millions of mobile devices while allowing multiple independent GSM networks within the same area. GSM also introduced the popular

[13] http://www.openmobileis.org
[14] http://equip.sourceforge.net
[15] Signal reflexions on objects such as buildings and system-independent jamming sources.
[16] The same frequency can be used within different GSM cells as long as the distance between the corresponding antennas is high enough

Short Message Service (SMS) as additional asynchronous communication infrastructure. The GSM standard works with a data rate of 9.6 kBit/s and includes several frequency domains with different channel numbers, as well as one *Uplink* and one *Downlink* for each domain. The *GSM900* and the *GSM1800* domains are often used, whereby GSM900 occupies the 890-915 MHz frequency band as uplink and the 935-960 MHz frequency band as downlink. GSM1800 uses 1710-1785 MHz as uplink and 1805-1880 MHz as downlink [Gut07]. Country-wide wireless base stations, *Handover* and *Roaming* functionality between stations, and the detection and suppression of jamming sources, are supporting a reliable mobile communication in the entire country and also abroad. The GSM standard is the infrastructural base for more advanced cellular network concepts like *HSCSD*, *GPRS*, and *EDGE*, also referred to as *2.5G* networks (see section *GSM Extensions* [3.3.1.2]), as well as for next generation networks (see section *UMTS* [3.3.1.3]).

The GSM architecture consists of three subsystems. The *Operation and Maintenance Subsystem (OMSS)* is used by the operator for administration and controlling purposes. For the management of the network internal user data and for the accessability to external networks, the *Mobile Switching and Management Subsystem (SMSS)* has been specified. Finally, the *Base Station Subsystem (BSS)* is responsible for accessing the actual *Mobile Stations (MS)*[17].

The communication between an MS and the GSM network is performed through the air interface[18] of the *Base Transceiver Station (BTS)* related to a distinct GSM cell. Due to economic reasons, BTS clusters are controlled by a *Base Station Controller (BSC)*, which is furthermore under the care of the *Mobile Switching Center (MSC)*. Data exchange to foreign networks goes through the *Gateway Mobile Switching Center (GMSC)* as well as through the *International Switching Center (ISC)*. The network management is the reponsibility of the *Operation and Maintenance Center (OMC)*. Multiple GSM databases are available to manage the users-centric data: *Equipment Identifier Register (EIR)*, *Authentication Center (AUC)*, *Home Location Register (HLR)*, and *Visitors Location Register (VLR)* [Rot02]. Figure 3.4 illustrates the introduced compontents and their relationship to each other. The specific GSM components are not addressed in more detail, since this does not lie within the focus of this master's thesis.

Each user-related *Subscriber Identity Module (SIM)* contains a globally unique *International Mobile Subscriber Identity (IMSI)*[19] which must be readable by the MS before a user is allowed to log into a particular GSM network. Also the built-in *International Mobile Station Equipment Identity (IMEI)*[20] of the used MS is determined by the provider to be able to identify the behaviour[21] of a connected mobile device.

3.3.1.2. GSM Extensions

The *High Speed Circuit Switching Devices (HSCSD)* technology is an early improvement of the GSM capabilities, which became too limited for some applications. Through a better data encoding and a multiplexing of several channels, HSCSD increases the theoretical data rate to 115.2 kBit/s under adoptable setup costs. Used a pure GSM-compliant handset, the end users had to exchange their devices by an HSCSD enabled one. As it is the case with GSM, HSCSD

[17] GSM terminology - all mobile devices related to the GSM standard.
[18] The remaining network component communication infrastructure is usually based on fiber optics.
[19] Stored in the provider's HLR.
[20] Stored in the provider's EIR.
[21] The IMEI's are tracked in lists to identifiy currently used, stolen, or outdated devices.

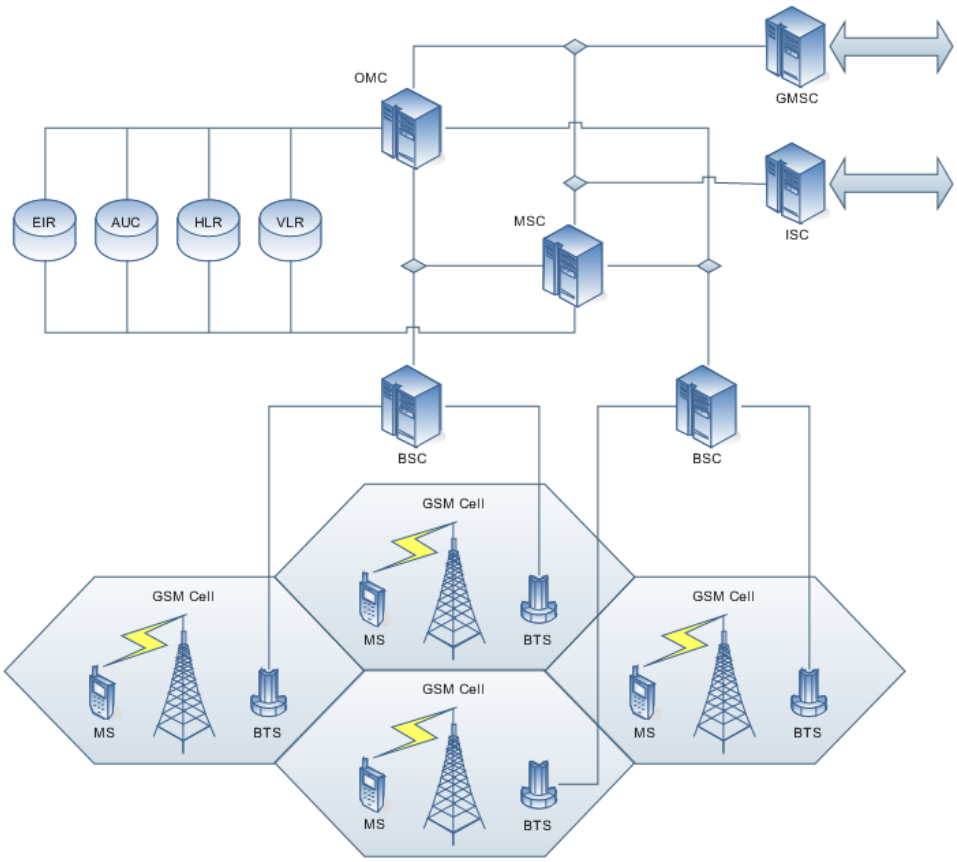


Figure 3.4.: *Relationship between GSM Network Components*

also is a *Circuit-Switched* technique, which means that the end user has to pay for a service as long as a connection is established. This behaviour is not suitable for mobile web connections, which are working in a dialog-oriented manner.

To allow a smooth accessability to *Packet-Oriented* networks[22] within the mobile domain and a better utilization of network resources for data-oriented services, the *General Packet Radio Service (GPRS)* standard was released. It also introduced the *Multimedia Message Service (MMS)* based on WAP 2.0 implementations (see section *HTTP Support* [3.3.3.1]). GPRS is available in Austria since the turn of the millenium, which theoretically enables services with a speed of 171.2 kBit/s under optimal conditions and the usage of eight multiplexed transfer channels. Due to the wide distribution of circuit-switched communication systems within pre-GPRS networks, the additional cost for corresponding network infrastructure extensions was high and similar to the introduction of the HSCSD standard, new end devices were necessary. GPRS devices are classified according to their capability to manage voice- and data-oriented services[23]. One important category of components within the new infrastructure are *GPRS Support Nodes (GSN)*, which are acting as packet routers and gateways to external networks. The GPRS specification allows registered mobile end devices to be *always online*, while the resources are only claimed when data packets have to be exchanged [Rot02]. GPRS-based services are either of *Point-to-Point (PTP)* or of *Point-to-Multipoint (PTM)* nature and fully support the *TCP/IP* protocol stack [Umt07].

[22] E.g. IP and X.25 based networks.

[23] E.g. *Class A* allows both services in parallel, *Class B* uses the services in a mutually exclusive manner, and *Class C* devices have to switch the services manually

The last step towards 3G networks is the *Enhanced Data Rates for GSM evolution (EDGE)* extension. An enhanced modulation strategy named *8 Phase Shift Keying (8-PSK)* allows a transmission of three bits per cycle, resulting in a theoretical data rate of 473.6 kBit/s when using eigth multiplexed transfer channels[24]. The problem with EDGE is the higher propability of data transfer errors in conjunction with the 8-PSK modulation. To ensure a proper service quality with EDGE, the standard modulation technique of GSM called *Gaussian Minimum Shift Keying (GMSK)* is used if the error rate is too high. EDGE can be combined with HSCSD and GPRS resulting in network services with the acronyms *ECSD* and *EGPRS*, whereby their setup also requires an extensive modification of the existing infrastructure.

3.3.1.3. UMTS

The *Universal Mobile Telecommunication System (UMTS)* is one of the *Third Generation (3G)* mobile network standards. 3G networks are designed to meet the requirements of contemporary and future mobile applications by a better utilization of available frequencies and higher data rates. UMTS is mainly used as an european system today, whereby its specification was initiated in the 1990s by the *International Telecommunication Union (ITU)* through an appeal named *International Mobile Telecommunication 2000 (IMT-2000)*[25]. The intention of ITU was the establishment of an internationally used communication standard for mobile devices[26]. At this time an expert group called *3rd Generation Partnership Project (3GPP)* was formed to promote the standardization of 3G technologies. UMTS with the according air interface *UMTS Terrestrial Radio Access (UTRA)* was a suggestion of the ETSI and reached its final stage in Mai 2000 [Rot02]. UTRA is the core element of the UMTS infrastructure and specifies two wireless radio interfaces: UTRA *Frequence Division Duplex (FDD)* with an uplink frequency band of 1920-1980 MHz and a downlink of 2110-2170 MHz, UTRA *Time Division Duplex (TDD)* with an uplink frequency band of 1900-1920 MHz and a downlink of 2010-2025 MHz. The total auction sales for the corresponding country-specific and limited UMTS frequency licences have been enormous[27]. Operators without a licence may use EDGE as an alternative, which provides the best approximation, but with a worse network utilization compared to UMTS [Umt07].

As a logical specification conclusion, the network technology is mainly based on GSM. The network adaption is however very cost-intensive due to the setup of the 3G UTRA interface, which is completely different compared to the installed 2G interface. The new standard is designed to support much higher bandwidths and various usage scenarios. It is therefore more than a simple re-definition of the air interface. A focus lies on mobile IP-based multimedia services with data rates up to 2000 kbit/s, on QoS configuration possibilities[28], as well as on a radio access to multiple infrastructures including satellite systems and cordless phones[29]. UMTS application scenarios are: *Internet Services* (world wide web, news, booking, etc.), *Entertainment* (e-books, video clips, etc.), *Location-based Services* (logistic, navigation, location determination, etc.), *Financal Services* (online banking, shopping, etc.), *Communication* (voice telephony, video telephony, e-mail, greeting cards, etc.)

[24] In reality data rates up to 170 kBit/s are possible.
[25] 2000 refers to the year of the planned introduction and the used operating frequency of 2000 MHz.
[26] This goal was not reached and a variety of 3G networks are under usage today. In the USA for instance the 3G network *CDMA2000* was integrated into the existing 2G *IS-95* mobile network [Umt07].
[27] 50.8 billion euros [Rot02] (milliard under european conditions).
[28] Defined QoS categories: *Conventional*, *Streaming*, *Interactive*, *Background*
[29] Especially phones based on the popular *Digital Enhanced Cordless Telecommunications (DECT)* standard.

The UMTS reference architecture is categorized into different domains. The two major ones are: *User Equipment* and *Infrastructure*. The user equipment domain includes the mobile devices and is divided into the *User Services Identity Module (USIM)*[30], which manages service-specific user information and the *Mobile Equipment* domain. The infrastructure domain consists of the *Access Network*, which is usually the *UTRA Network (UTRAN)* and the *Core Network*. This core network includes the *Serving Network* responsible for the circuit-switched and packet-switched data transfer and location-based functionality. The *Home Network* domain is a further component of the core network and includes services, which are not depending on the user's current location. As a last domain within the core network, the *Transit Network* domain covers services needed for cross-network communication [Rot02]. Figure 3.5 represents this architecture.

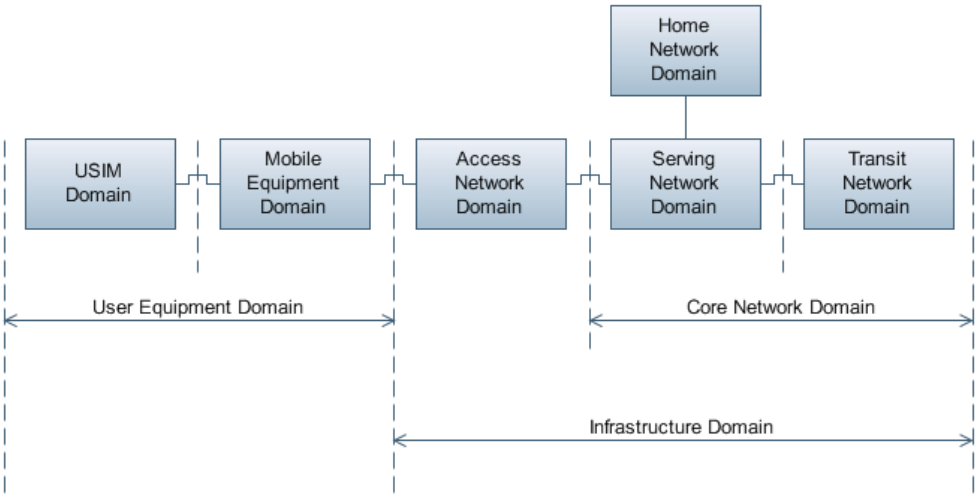


Figure 3.5.: *High-level UMTS Architecture*

In future releases of the UMTS specification the *All IP-Network* will be propagated, providing each service (including voice telephony) over the *IPv6* standard. With IPv6 each mobile device will also be individually addressable.

3.3.2. Symbian Communication Infrastructure

Symbian's most important communication servers are the *Communication Database Server*, the *Serial Communications Server*, the *Telephony Server*, the *Socket Server*, and the *Host Resolver Server*. The Symbian client-server model is overlayed by a communication infrastructure, which is divided into four layers: *Physical Device*, *Device Driver*, *Protocol Implementation* and *Application*.

The Physical Device represents the hardware which is seen as an abstract layer by the system. As part of the second layer, the typical device-dependent *Physical Device Driver (PDD)* directly communicates with the hardware, whereby the *Logical Device Driver (LDD)* is responsible for buffering, flow control, etc. The Protocol Implementation contains four important modules. To be able to use serial port services, *CSY* plug-in modules[31] for the Serial Communication Server were defined. *TSY* modules are Telephony Server plug-in modules for voice-centric

[30] Implemented on the user's SIM card.
[31] at least *RS232*, *IrDA*, and *Bluetooth*

services. For supporting protocol services the Socket Server plug-in modules [32] named *PRT* were implemented. Finally *MTM* modules are message type modules. Amongst others, the telephony service as basic communication application is shipped with the device by default. Further client applications are developed by third parties. Individual clients and communication subsystems may access the *Communication Database (CommDb)*, which holds communication-related settings like *Internet Access Providers (IAPs)*, *Internet Service Provider (ISPs)*, GPRS, Modems, Locations, WAP settings, etc. Multiple IAPs can be active and the client/user can choose which one should be used for accessing a particular network. In the same way GPRS/UMTS QoS parameters may be configured.

The platform-optimized *TCP/IP Services* for mobile clients can be used from many network interface types and include the *Internet Protocol* (*IPv4*/*IPv6* dual stack), the *Internet Control Message Protocol (ICMP)*, the *User Datagram Protocol (UDP)*, and the *Transmission Control Protocol (TCP)*, as well as the *Domain Name Service (DNS)*. This OSI protocol stack implementation also considers security aspects like IPSec. The IP is part of the abstract *Packet Data Protocol (PDP)* used by external packet data networks to communicate with GPRS networks. A mobile wireless service call or session is represented by the PDP context, which can either be primary or secondary. A *primary* PDP context is a specific connection from the mobile device to a network with one assigned IP address. A *secondary* PDP context shares the IP address with the primary one, but it is seen as a different network connection [Sym05].

3.3.3. Generic Connectivity Framework

A mobile application developer has several possibilities to exchange packet-based information with network nodes on different levels. The *Generic Connectivity Framework (GCF)* is a Java-based approach for this purpose. The network APIs for JavaSE occupy more than 200kB of memory and they partly reference to further packages. Furthermore, optimized mobile implementations are more hardware-centric and more homogeneous than `java.net` packages. Due to this inconsistencies, the JCP specified the GCF[33] as a unified set of network APIs optimized for CLDC devices. All MIDP 2.0 terminals support at least two application layer protocols: *HyperText Transfer Protocol (HTTP)*[34] and *HTTP over TLS/SSL (HTTPS)*[35] [Sch04].

3.3.3.1. HTTP Support

The standard OSI TCP/IP stack is not working sufficiently related to wireless mobile connections, due to the higher error rates and the higher latency, intrinsic to wireless networks. For that reason, an alternative OSI-compliant protocol stack was specified and implemented on data-network enabled phones. An access to Internet resources over this *Wireless Application Protocol (WAP)* stack however requires an TCP/IP-aware *WAP Proxy* to the wired IP network. The main idea of WAP was to deliver web content to mobile devices, to be accessable through *Mobile Micro-Browsers*. However mobile web content browsing was and is due to very small screens and missing input capabilities not very user-friendly and a long latency of mobile networks as

[32] at least *TCP/IP*, *IrDA*, and *Bluetooth*
[33] `javax.microedition.io`
[34] In version 1.1
[35] *Transport Layer Security*/*Secure Socket Layer*

well as unmoderate service fees[36] are further decreasing the usability of the supposed mobile web experience (see section *Motivation - Mobile 2.0* [2.5.3]). In general *Mobile Widgets* should be favoured over web navigation paradigms, which typically integrates the mobile Internet more efficiently. Nevertheless, WAP is an adopted technology, and might reach the outstanding honor in prospective setups and scenarions.

The first release of WAP in 1998 defined a compact binary processing of the WAP-specific *Wireless Markup Language (WML)*[37], more suitable to be transferred to mobile clients over low bandwidth cellular networks. The translation between the compiled[38] and the plain text version of the web content is the task of the WAP proxy, as well as the syntactical analysis[39] of the WML page delivered by the HTTP server, the *Cookie* management and the domain name resolution[40]. Since HTTP and HTTPS are network-agnostic protocols, they can also be transported over the WAP-specific ones like *Wireless Session Protocol (WSP)*, *Wireless Transaction Protocol (WTP)*[41], and *Wireless Datagram Protocol (WDP)*.

With the introduction of *WAP 2.0* in 2002, the mobile Internet access has become more familiar, since WAP-related protocols and security concepts were mainly replaced by *wireless profiled* standard Internet communication protocols such as IP, TCP, HTTP[42], and SSL. WAP 2.0 communication specifications are working with forseeable and existing air interfaces and their bearers including GPRS and 3G network technologies [WAP02]. However, as introduced in *Symbian Communication Infrastructure* [3.3.2], an adapted connectivity model must be used, which also supports an WAP 1.0-independent TCP/IP service implementation for the mobile IT domain[43]. According to [WAP02], *'WAP 2.0 leverages IETF work in the Performance Implications of Link Characteristics (PILC) Working Group to develop a mobile profile of TCP for wireless links'*. Thanks to this new wireless protocol specification, web content can also be delivered in plain text *XHTML Mobile Profile (XHTML-MP)*. This step flattens the learning curve for mobile content creation and may by-pass an operator-specific WAP proxy by communicating directly with the original web server over HTTP/1.1. This means that an IP network connection can either be established through a carrier-based proxy[44] or an *Internet Access Point* [Vir05]. The usage of an intermediate network node may offer advantages in terms of optimized communication processes such as with location-based services and privacy concerns [WAP02]. A proxy is also necessary to be able to implement WAP 2.0 services like *WAP Push* functionality for real-time applications. As already introduced in section *Mobile Platforms - Symbian OS - Deployment* [3.1.2.6], the MIDP 2.0 implementation uses a *Public Key Infrastructure (PKI)* to be able to verify signed MIDlets. This PKI specification also considers WAP-specific security profiles, which enables *HTTPS* and *Secure Stream* connections for mobile devices. The mobilized version of HTTPS therefore also provides *Authentication*, *Confidentiality* and *Data Integrity* such as with standard HTTPS [Sch04]. The WAP 2.0 approach increases the bandwidth utilization and requires more intelligent web browsers, whereby a successful introduction is also bound to more powerful devices, both in screen resolution and color model as well as in CPU and memory metrics.

[36] In cases where a mobile Internet access is not based on a provider-specific flat rate, the service fees are usually quite high.
[37] Also referred to as WAP sites.
[38] MIME Type: *WML Compiled (WMLC)*
[39] which is usually a task of the browser
[40] http://dev.mobi
[41] A lightweight transaction protocol for thin clients, optimized for wireless datagram networks [WAP02].
[42] Also referred to as WP-HTTP. Provides message body compression and secure tunnels.
[43] At this time a traditional WAP stack as well as a TCP/IP stack are usually implemented on smart phones. The stacks are dynamically switched depending on the web content.
[44] The usage of non-standard HTTP ports may sometimes cause problems.

3.3.3.2. Low Level Network Support

The GCF also offers low level network support in form of APIs for the transport layer protocols TCP and UDP, which are not mandatorily implemented on devices. This MIDP 2.0 Java APIs allows the development of enhanced mobile side functionality like servers on the basis of *Socket*[45] connections, *Serial Communication* services, and *Push Registry* services. The Push Registry functionality must not be confused with WAP Push, which is a part of the WAP 2.0 service infrastructure and related to the mobile browsing paradigm. The MIDP-based approach is quite similar to the UNIX-based *Internet Super-Server (inetd)*. The Push Registry service is started at boot time and used to process incoming connection requests by starting the requested services, encapsulated into MIDlets. This task is performed by the AMS [Sch04].

3.3.4. Bluetooth

The Bluetooth (BT)[46] technology was specified in 2001 as a data- and voice centered, universal short-range and low power information carrier which is now integrated in many handhelds for establishing wireless *Personal Area Networks* (PAN), also called BT *Pico Networks*, with surrounding Bluetooth-enabled hosts [Hol03b].

3.3.4.1. Working Principle

In contrast to IrDA[47], the data exchange is based on radio technology which allows an operating radius of up to 150 meters[48] and a maximal data rate of 2.1 MBit/s[49], without the necessity of having a line of sight between the communicating nodes. BT typically supports the pervasive computing paradigm, since it is a cheap technology which can be seamlessly integrated into many kinds of smart objects to establish *Ad-Hoc Networks* between them. The initiator of such a network is referred to as *Master* and the remaining nodes are called *Slaves*. A pico network may hold up to 7 active and 255 slaves in power-save mode, which might be activated on demand. Slaves in power-save mode are either in the *park*, *sniff*, or *hold* state, which distinguishes the way on how they participate on a session with the master. Communication links between overlapping pico networks are also specified through so called *Scatter Networks*. The BT technology uses the 2.4 GHz ISM-Band[50] which is available in most countries without a licence fee for low-power usage. The communication within such pico networks is based on *Frequency Hopping Spread Spectrum* (FHSS) which uses 79 channels of 1MHz each. Active transmission channels are changed arbitrarily at a rate of 1600Hz, whereby a robust communication link is ensured. This technique requires a shared knowledge between nodes about the starting point of the frequency hopping algorithm which therefore introduces some kind of secure links as well [Hol03a]. According to the IEEE 802 standard, each BT device is associated with a worldwide unique manufacturer-specific 48 bit address, which is referred to as the *Medium Access Control* (MAC) address [Hol03a].

[45] In UNIX terms a socket describes a programming interface for several transport protocols. In MIDP terms a socket only supports TCP.
[46] Ericsson named this technology after the norwegian viking king *Harald Blåtand*.
[47] Infrared Data Association.
[48] For class 1 devices with a transmission power of 100mW.
[49] According to the Enhanced Data Rate Specification of 2005 [Bia08].
[50] *I*ndustrial, *S*cientific, *M*edical Frequency Band

3.3.4.2. Protocol Stack

The most important protocols in the BT protocol stack are the *Logical Link Control and Adaption Protocol* (L2CAP) responsible of multiplexing high level protocols to an asynchronous connection, the *Service Discovery Protocol* (SDP), and the *Radio Frequency Emulation of the Serial COM Ports* (RFCOMM), which emulates the RS-232 protocol over the L2CAP channel [Hol03b]. The L2CAP protocol is part of the *Connection Management and Control* Layer within the BT stack. This layer is situated on top of the *Baseband* layer, which provides mechanisms for the connection establishment, and the *Radio* layer, which performs the signal modulation and the signal transmission over the air. To ensure an interoperability between BT applications of different vendors the *Bluetooth Special Interest Group* (BT SIG) defines several profiles which might be seen as vertical layers covering different protocols of the stack. The most important profiles are the *Generic Access Profile*, the *Service Discovery Application Profile*, the *Serial Port Profile*, and the *Generic Object Exchange Profile*.

The common network protocols UDP, TCP, IP, and PPP are also defined in the BT stack, whereby the TCP/IP over PPP functionality is also implemented as a more efficiently alternative through the *Bluetooth Network Encapsulation Protocol* (BNEP). A further important part of the BT stack is the *Object Exchange Protocol* (OBEX) which originally was specified for IrDA to exchange data objects between short range wireless interfaces. OBEX has similar characteristics as the HTTP protocol, however the headers are not mandatory in use, but may be used to describe the transfered byte arrays or byte sequences. During an OBEX session the client can `GET` objects from an OBEX server or `PUT` objects to it. The objects, which are either transferred as a whole or as successive packets, offer a developer a convenient mechanism to exchange multimedia objects, *vCards*, or *vNotes* with remote BT nodes running an OBEX server. The objects itself are constructed within MIDlets [New08]. With the *OBEX File Transfer Profile* it is possible for remote applications to browse and manipulate files directly on the device. An open-source implementation of the BT Stack called *BlueZ* is integrated into the Linux kernel.

3.3.4.3. Service Implementation

MIDP implementations provide the `javax.bluetooth` and `javax.obex` APIs through the optional *JSR-82* package to access the bluetooth functionality of a device. The two APIs are split into three functional categories. The *Discovery* category covers methods for browsing devices and their services in range, as well as methods for device registration purposes. Once a Bluetooth enabled application is executed it registers its service records within its own *Service Discovery Database* (SDDB), to offer them for searching or browsing by inquiring SDP clients through the local SDP server. Each service is referred to a 128 Bit *Universally Unique Identifier* (UUID). Sent packets within a pico network have the same 72 Bit *Access Code*, which is checked by the receiving node before the payload is fetched. As one particular code category, the *Inquiry Access Code* (IAC) is used to identify surrounding devices, whereas the *General Inquiry Access Code* (GIAC) and the *Dedicated Inquiry Access Code* (DIAC) subcategories are used to perform a search either without or with restrictions according to certain device characteristics [Hol03b]. The *Communication* category provides interfaces for exchanging data between discovered devices, and the *Device Management* category for controlling established connections. The APIs also interfaces with the *Bluetooth Control Center* (BCC) of the device to manage secure connections between nodes [New08].

3.3.4.4. Security

To be able to authenticate devices within an ad-hoc network, a *Challenge-Response* dialog may be initiated either by the master or by a slave, which are further refered to as the *Verifier* and the *Claimant*. The verifier sends a random number[51] to the claimant. The queried device creates a 128 Bit *Combination Key* with the received as well as an own random number, its MAC address, and a shared secrete passkey[52]. If the responsed calculated key is accepted by the verifier, the devices are said to be *Paired* and a circuit- or packet centric communication can be performed trustworthy without a further user intervention, even if both devices are locked for discovery by the user. This authentication mechanism is single-edged, however it may be performed from the claimant in the same way. Furthermore the transferred data might be encrypted by the master, which is however not supported by each BT device and typically relates on higher level protocols. Even if the BT security standards are quite high, it might be still possible for unauthorized persons to access private data on remote devices or even to send messages from hijacked devices via a BT link. According to [Bia08] hackers may use software which lets devices react to discover inquiries even if they are not configured in this way. Further security problems may also be based on erroneous service implementations by device manufactures themselves, which is frequently the case for particular models.

3.4. Service-Oriented Computing

In this section software systems are discussed under the viewpoint of adaptivity, re-usablity, management and inter-domain communication with a focus on mobile environments. Within modern and large-scale B2B and B2C scenarios, mobile applications can be seen as independent and distributed software components with a high degree of interconnectivity with numerious related remote counterparts, as their common characteristics. Interconnectivity in this sense represents a method to define and reach communication targets more efficiently and effectively.

3.4.1. Infrastructures

3.4.1.1. Enterprise Services

Many *Business-2-Business* (B2B) collaborations are built in the scope of *Enterprise Application Integration* (EAI) and automatization of business processes. An important fact in this relation is an adequate control of the workflow between heterogeneous systems, which is usually supported by EAI tools. Within the enterprise domain suitable conceptional and organizational strategies in the software development process are important for a long-term success of a product. The hyped *Service Oriented Architecture* (SOA) is an accepted and suitable concept to reach this goal. An SOA reduces the former complex workflow handling to simple remote services invocations. Web Services are one way to build a SOA, whereas the integration of heterogeneous mobile

[51] For a secure transmission of this number a *Initialization Key* is used, which can be calculated from a public random number, a MAC address, and a passkey [Hol03b].

[52] A shared *Link Key* is generated from the passkey which is stored in the device's persistent memory and used for a transparent authentication later on. If both parties have a fixed passkey, no authentication is possible. Sometimes devices have a default passkey of `0000`.

devices as agile and interactive components within business solutions is due to the availability of mobile web service frameworks more attractive than ever before.

Even if the SOA strategy has been heavily researched and advanced in the last 10 years and the advantages are obvious, the degree of deployment within real-world software systems is quite low [Bec08]. The US *Nucleus Research Center*[53] recently finalized an IT market study, which brought to light that only 27% of IT projects are somehow based on a SOA right now. Only 40% of the developers are using SOA concepts, which is furthermore often concentrated on a few web services. The reasons for the stagnation may be seen in the avoidance of an exhausting refactoring of existing and proven products and a resulting production drop-out in correlation with high investments and a possible negative *Return on Investment* (ROI) for the final product. According to [Bec08], about 75-95% of the companies' IT budget is used to maintain and optimize products, whereas no resources are left to develop new systems from the scratch. Also the change to standard software solutions would mainly depreciate the former work while negating competitive advantages reached through a successful software release. On the other side, IT experts are aware of the additional investment to maintain large software systems with structural lacks, which often become visible if they are under frequent modification and their functionality and complexity is growing. Also ad-hoc decisions, sufficient in-house knowledge, euphoria of successful steps during the implementation process, and a thinking in terms of workarounds often lead to hardly maintainable systems at a later point in time [KBS05].

Existing systems are not seldom based on inflexible and closed mainframe applications written in *Common Business Oriented Language* (COBOL). An integration or a data synchronization with web-based or mobile terminals is often difficult to realize for this type of legacy systems. A desired process may be to easily transfer legacy software in a new open context based on a language-, technology-, and system-independent SOA with the help of some kind of *Enterprise IT Renovation Roadmap* [KBS05]. The main challenge in a subsequent system adaption lies within a liberation of a proven business logic with complex cross-dependencies towards functional service interfaces, especially if both, core and peripheral components, are affected. A trend therefore goes to the automatization of the refactoring step while preventing changes of the existing code whenever possible [Bec08]. Such mapping products are already available and might have the potential to encourage the SOA paradigm within the IT market by following the architecture's strategy of embracing existing and upcoming technologies.

3.4.1.2. Mobilized Services

Comprehensive mobility in the IT domain also means a requirement for mobile services, which should be available across time and space as well as on every device. Mobile properties[54] depend on the architecture on which the services are based. Mobile services may be coarsely classified in *Service Logic* and *Service Content*. In early telecom systems services have been mainly embedded in hardware components within the provider's internal infrastructure. This lack of accessability from an outside domain through independent software developers led to a dominance of the provider in the mobile IT market and has been a hindrance for developing flexible services [JDvT04]. Step by step services have been decoupled in form of third party service providers, which are operating on a specialized domain like SMS, WAP, or enterprise services. Furthermore 3GPP introduced the *Virtual Home Environment* for 3G mobile systems

[53] http://nucleusresearch.com

[54] Personalized service properties may be *passive* (e.g. look and feel), *active* (service behavior), or *content related* (user generated content, service inquiry results).

to guarantee flexible access to subscribed user services from home-, as well as from foreign networks. Nowadays, the dependence from a particular operator when developing mobile services, is based on the involved platforms. WAP and JavaME services, for instance, do not require special agreements, whereas SMS services do so. Provider-related services have many advantages, which are often difficult to implement in an independent way. For instance, a determination of the user's current location through a cellular network might be helpful in relation with a location-based information system during the absence of satellite links. A typical problem with this approach is that the provider often uses an unsecure transmission channel for carrying sensible user data. On the other side, network providers usually have the status of a confidential partner for the customer [Cha06].

[JDvT04] sees an ultimative goal of a further service decoupling in an open and *'...ubiquitous access to data services ... no matter where the service content and service logic is located'*. In this sense, a novel concept is the composite service, which is spanning over several domains including home networks, foreign networks, Internet, enterprise networks, and PANs, whereas a user's profile including privacy preferences is distributed or replicated across these networks. The implementation of such universal services however requires a sophisticated interaction design, security and usability roadmap as well as a suitable inter-domain communication infrastructure (see section *Umiquitous Mobility* [2.5]).

Network operators already started to support this concept by opening interfaces to Internet based services on business as well as on entertainment level. The operator's technical environment however typically consists of proprietary and open distributed systems such as n-tiered enterprise application platforms like *JavaEE*, *.NET*, or CORBA. By default, these infrastructures do often not support a loosely coupled interaction in SOA style. Two standardization efforts try to improve this situation: *OMA Web Service Enabler Release* (OWSER)[55] and *Parlay X Web Services* [56] [Cha06]. With a focus on such standardizations, *'operators can leverage Service Composition to reuse their existing legacy systems and Services to build higher level services for internal deployment and expose it as services to enable new innovations on existing assets'*. Furthermore, *'web services will play a major role in making open IT solutions available in Mobile Environments.'* [Cha06].

3.4.2. Evolution of the Service Concept

The term service is used in many different ways within the IT industry. According to [KBS05] a service *'...denotes some meaningful activity that a computer performs on request of another computer program'*. In a more technical sense, [KBS05] defines a service as a *'...remotely accessable, self contained application module'*, whereas the communicating nodes are referred to as a *Service Consumer* and *Service Provider*. It is important to understand that a SOA is no middleware and therefore no concrete technology, but a description of a structure which encapsulates activity of a certain complexity on different levels of abstraction. The service model is a quite old concept which found its way towards efficient inter-domain communication patterns within the last decade. The idea of service orientation arose from *Programming Paradigms*, *Distributed Technologies*, and *Business Computing* [KBS05]. The business domain is the strongest power in the SOA evolution, since its abstract components facilitate an understanding and a shared knowledge of processes and data flows on both sides, technology and business.

[55] http://www.openmobilealliance.org/technical/release_program/owser_v10.aspx
[56] http://www.parlay.org/en/specifications/index.asp

3.4.2.1. Programming Paradigms

One of the most popular languages for computer science was *Pascal*, introduced by Nikolaus Wirth in 1970. It was an early functional programming concept which provided some form of abstraction and software libraries. However, since multi-purpose functions were difficult to create and many parameters had to be passed, the trend went further to software modules and components whereas the term *Encapsulation* got popular in the '80s. In those years *Modula*, *ADA*, and *Prolog* were created. With the mainstream usage of the *Object-Oriented Paradigm* (OOP)[57] through languages like *Smalltalk*, *C++*, and *Java*, the concept of individual entities in form of objects and the *Inheritance* of functionality have been added. Important software patterns like *Interfaces* were developed from OOP. The granularity and abstraction of OOP implementations are however to fine for the service concept [KBS05].

3.4.2.2. Distributed Computing

Calling remote processes on physically distributed machines residing in multiple networks in a seamless and controlled way is an essential service characteristic. In early distributed systems the computing power wasn't distributed, but the data I/O in form of remote *Terminals*, while the logic still resides on huge and expensive mainframes. During two research projects at the University of Berkeley *UNIX* OS was developed, whereas the distributed computing concepts had been mapped on small and powerful workstations in the '70s. An easy remote access to a particular host over a network with tools like *Telnet* and a transparent provisioning of storage space through the *Network File System* (NFS)[58] have been introduced in 1984 [KBS05]. As soon as relational *Database Systems* (DBS) became available and companies adopted the *Client-Server* pattern, *Stored Procedures* were used to conduct functions directly within the DBS.

The next step was to blur the border between the client and the server by introducing *Distributed Middleware* models in *Remote Procedure Call* (RPC) style as a solution for *Application Heterogenity*, with the Internet as underlying platform [TvS02]. Later on, a combination of a distributed computing environment and the OOP concept, resulted in the specification of the *Common Object Request Broker Architecture* (CORBA). The former structure of a server, which offered a large number of functions was broken down into uniquely identificable and remotely accessable objects managing their own state. Due to the involved *Interface Definition Language* (IDL), CORBA objects are language-independent and registerable within a naming service, which enables a runtime discovery of the available server objects. Furthermore, this strategy is mapping the OO design to the network and follows the *Interoperable Object Reference* concept which comes along with *Object Request Brokers* (ORBs)[59] for accessing and manipulating remote objects. To ensure interoperability accross vendor boundaries the TCP-based *Internet Inter-ORB Protocol* (IIOP) was also introduced at this time [Nok06]. CORBA components are executed in *CORBA Component Model* (CCM) containers within application servers and are mainly used in telecommunication and financial systems. However the architecture's structure is often too fine-grained and too complex for the majority of enterprise applications. As a result, clusters of server-side objects in form of more abstract middleware systems have been

[57] The concept was first realized in 1967 with the language SIMULA.
[58] Developed by Sun Microsystems.
[59] Further ORBs are *Remote Method Invocation* (RMI) from Sun and the *Component Object Model* family (COM/D-COM) from Microsoft, which are however bound to a particular programming language.

propagated in the '90s. Such a strategy got popular in 1997 [KBS05] through the *Enterprise Java Beans* (EJB) specification as part of the JavaEE platform. The EJB technology involves a sophisticated application container, which enables web services and manages application resources, including a naming service as well as extensive security features. Next to the EJB many other middleware solutions like Microsoft's *COM+* and *.NET*[60] technologies are available on the IT market at the moment, causing an ironic problem: *Middleware Herogenity*.

A solution for this problematic trend is the *eXtensible Markup Language* (XML) as *'...the smallest common denominator upon which the IT industry could agree'* [KBS05]. Since XML is independent of any middleware standard and does not require a heavy-weight infrastructure, it seems to be the ideal format for exchanging data between applications. Moreover, tools for processing and managing XML data such as the *Simple API for XML Processing* (SAX) and the *Document Object Model* (DOM) API are available on many different platforms. To be able to use this self-descriptive protocol efficiently within the enterprise domain as a higher level messaging format, more complex XML-based data types had to be specified. With a focus on the Internet as the main data carrier, DevelopMentor, IBM, and Microsoft [FTW07] invented the *Simple Object Access Protocol* (SOAP) as a suitable document-centric protocol for this purpose. This specification started the age of XML-based *Web Service* technology in 1998. SOAP was initially defined with respect to the HTTP protocol as the underlying transport protocol to make use of world-wide established and configured network infrastructures[61]. Due to its growing importance within IT systems, this extensible[62] communication protocol became a W3C recommendation in 2001. Microsoft also invented the *Web Service Discription Language* (WSDL) in anology to the IDL of CORBA, which also enabled various bindings to lower-level communication protocols of existing middleware systems like the *Message Oriented Middleware* (MOM). The most flexible approach to discover available services is through a publicly accessable service broker, which holds WSDL documents and according provider information (c.f. yellow pages). This registry is standardized through the *Universal Description, Discovery and Integration* (UDDI). The development of distributed computing architectures like RPCs, ORBs, EJB, MOM, and XML Web Services can be seen as communication models or middleware busses with different characteristics[63] as well as basis for the SOA concept.

As it is generally known, middleware layers are situated between the network software or the OS respectively and the user applications, which shield the developer from dealing with low-level functions. To integrate with several middleware systems, an additional layer of abstraction in form of technology-dependent *Adapters* between applications and the middleware busses are used. A suitable middleware interaction like illustrated in figure 3.6 increases the flexibility, interoperability, portability, and maintainability of distributed applications.

Even if standardized and proven middleware systems give developers more freedom, it is important to understand the concept behind the scene, to be able to choose and adapt technology for a particular problem and to handle potential conflicts.

[60] In contrast to JavaEE, .NET supports multiple component languages like C#, C++, and Java. However .NET is bound to Microsoft platforms. Currently the *Mono* project supports the .NET concept under Linux distributions.

[61] IIOP on the other side needs a special configuration of the firewalls.

[62] Unlike as with unflexible documents like HTML the developer is capable of introducing new application-specific tags. According to [FTW07] this fact mainly led to the success of SOAP.

[63] E.g. *Degree of Coupling* between components, *Asynchron* vs. *Synchron* data exchange, *Interface* vs. *Payload* semantics, etc.

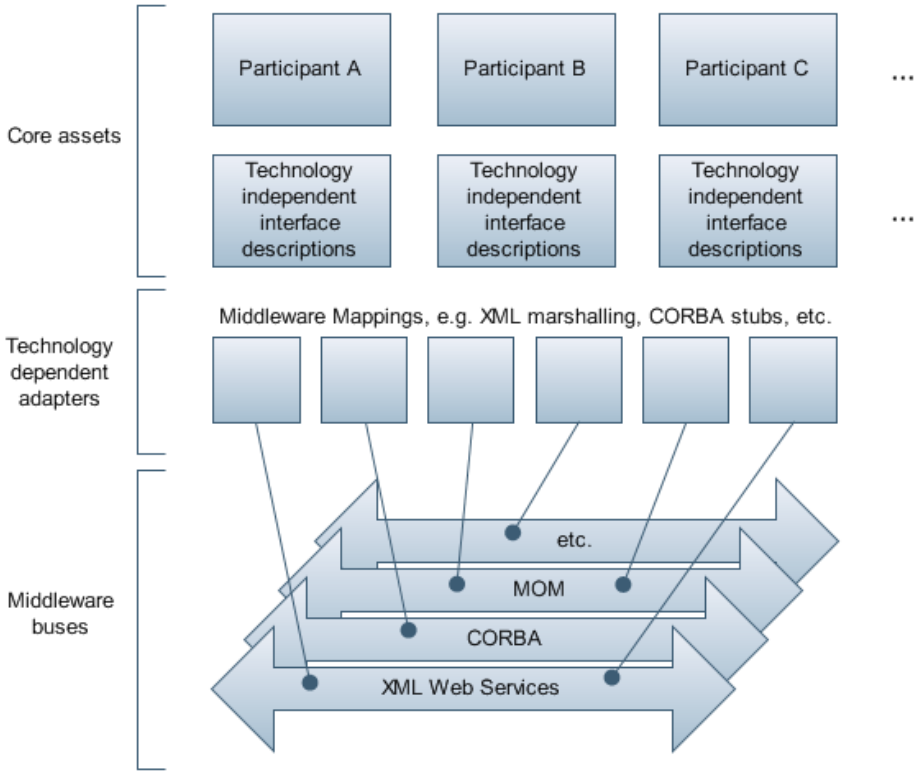


Figure 3.6.: *Connecting to Middleware Buses through Adapters*

3.4.2.3. Business Computing

The IT-based management of business processes is the source for the content in form of business data and business logic within software systems, and therefore as important as the underlying technical infrastructure in the evolution towards an SOA. In mainframe-eras and later on COBOL and FORTRAN have been suitable languages to implement solutions for business problems. In correlation with database systems the *Structured Query Language* (SQL) got very popular, which was mainly built for business analysts and not for database programmers. In 1981 *SAP* set a milestone within the business domain as they released the *R/2* system, suitable for enterprise-wide real-time processing of financial data and resource planning issues. Changes in the worldwide business market later on, led to more complex enterprise systems, as *Enterprise Resource Planning* (ERP) and *Supply Chain Management* (SCM). In the 1990ies the high availability and co-existence of enterprise software caused many problems and changed the business models once again towards *Enterprise Application Integration* (EAI). In the context of Internet networks and through the demand to integrate across organizational boundaries, a flexible and distributed service oriented architecture is nowadays the preferred business solution [KBS05].

The introduction and application of the SOA paradigm changed the enterprise computing domain significantly. Moreover, *'Similar to object orientation, which today seems to present the endpoint in the development of programming concepts, service orientation is the result of a long evolutionary process and has the potential to finally provide some stability in an environment that has been constantly evolving over the past 30 years'.* [KBS05]

3.4.3. Payload Communication Strategies

Next to *Web Services*, there are two other kinds of integration techniques concerning the participation of mobile devices: *Socket Connections*, and *Messaging Techniques*. When using sockets, one is fully responsible of the data structure to be transfered. This also requires a detailed knowledge of network programming concepts in multiple languages, as within heterogonous setups. If a software system does not need to transfer complex data structures such as serialized objects, one can define individual communication formats with socket connections. This is the most efficient solution in respect to message overhead and transmission speed. Since, this proprietary approach does not scale well, also a specialized understanding of how to access communication endpoints is needed. Messaging software typically occupies a lot of resources and slows down the system, and it is often hard to manage in terms of security issues.

Web Services offer simple invocations and less programming effort, however this mechanism is still incapable of fulfilling critical real-time processing requirements due to the costs of XML parsing and exchanging SOAP messages. On the other side, XML is very flexible in structuring data and basically well supported across platforms. With a focus on standardized communication techniques, the following sections outline the most common messaging methods for XML-based web services.

3.4.3.1. REST

The term *Representational State Transfer* (REST) describes an *Architectural Style* on networking systems, which arose from a Ph.D. dissertation[64] by Roy Fielding[65]. In this work the *Web* is seen as a *'...network of web pages (a virtual state machine), where the user progresses through an application by selecting links (state transitions), resulting in the next page (representing the next state of the application) being transferred to the user and rendered for their use'* [Res08]. A web resource is everything which is accessable through an URI, not only web pages: e.g. CGI scripts, images, servlets, XML documents, etc. One motivation for this work was to capture the evolution of the web, but it is also a guideline of how the web as a globally distributed application should work.

REST is no product, standard or protocol, however it may be used to design web services in a RESTful[66] fashion. Many search engines, shops, etc. are available as RESTful web services, even if this has not been the intended design goal. As their generic interfaces, resources are accessable and editable through HTTP `GET`, `POST`, `PUT`, and `DELETE` commands [Res02]. The result of a response from an REST application is typically a resource in form of an XML document holding some useful and structured data, which has to be processed and represented by the client application in some way. As soon as the content of an XML document contains a hyperlink mapping (E.g. through `XLink` tags), a client may be able to traverse into deep nested and loosely coupled resources through one initial URI. As a system logon feature, HTTP Basic Authentication and an SSL enabled channel might be used.

[64] http://www.ics.uci.edu/~fielding/pubs/dissertation/top.htm

[65] Former director of the Apache Software Foundation and initiator of Apache HTTP server. - http://roy.gbiv.com/vita.html

[66] Within REST systems a service has a different meaning since the focus is on the resource, not on service implementations.

In contrast to SOAP and XML-RPC, REST systems can be configured easily by system administrators through firewall, proxy, and server settings by enabling or disabling resources. RESTful web services *scale* with the web and its infrastructure, whereby a widespreaded and flexible inter-domain resource handling (resource formats can be easily appended through MIME types) through *stateless* operations is one of their biggest advantages. Furthermore, REST supports a caching of service operations through standardized HTTP GET caching methods. The core of the REST server is working with the same principles as an RPC server. The only difference is that an HTTP interface is used instead of a component-base one [Res02]. As with SOAP and XML-RPC, REST is not suitable for an efficient local communication between components like the application server and a database system. Also the serialization and deserialization between objects and their XML representations requires XML binding frameworks. Unlike middleware systems, RESTful web services can be seen as *pure* web services since they are only using established web technologies like HTTP, URI, MIME types, and global addressing. RESTful web services are widely available and well supported by common web servers.

3.4.3.2. XML-RPC

The *Remote Procedure Call* (RPC) technology was introduced by Sun Microsystems in the 80ies[67]. Within RPC systems *'...the syntax and the semantics of remote calls remain the same whether or not the client and server are located on the same system'* [KBS05]. Even if an implementation of this technique is not related to an RPC-conform RFC protocol, the communication infrastructure is said to work in RPC-style. The most instances are based on a technology-intrinsic synchronous communication. Due to various competing RPC implementations in the 80ies, the *Distributed Computing Environment* (DCE) standard tried to overcome this situation, however with less success. As a concrete instance of this style, the XML-RPC payload protocol is widely used for XML-based middleware systems, even if it is no official standard [Nok06]. XML-RPC is faster and easier to understand as the related SOAP protocol, because it is less customizable and extensible. For instance this lightweight protocol does not support *Namespaces*, whereby the same processing order must be maintained for wrapping and unwrapping the message content.

3.4.3.3. SOAP

The *Simple Object Access Protocol* (SOAP)[68] was designed in a very flexible way. In fact, it is a communication protocol for remote invocations, a standard for interoperability, a protocol for exchanging documents, and a business communication language [CJ02]. Unlike as the original acronym indicates, handling SOAP data is in principle as complex as handling with the involved *XML Schemas*, however much of that complexity is managed by corresponding frameworks and tools.

In contrast to version 1.1, the current SOAP 1.2 specification[69] describes a whole messaging framework instead of a single protocol [FTW07]. However, it still defines an one-way protocol,

[67] E.g. NFS(RFC 1094) was implemented with respect to *SUN RPC* [KBS05].

[68] The relation to an *Object* originally comes from invocations of Microsoft's *COM Objects* over the Internet [CJ02]. Today SOAP is usually used within a non-OO context, whereas the SOAP 1.2 specification declared SOAP as a simple name instead of an acronym [FTW07].

[69] http://www.w3.org/TR/soap12-part1/

whereas according messages are often combined to reach a synchron as well as an asynchron request-response schema. SOAP 1.2 is aware of namespaces and its specification includes formatting conventions for describing the content and routing information within an *SOAP Envelope*, a protocol binding, data type encoding rules, and a RPC mechanism. The new specification also introduced the term *Message Exchange Pattern* (MEP) to indicate two available communication scenarios: a dialog-oriented *Conversational Message Exchange* suitable for MOM infrastructures and the traditional XML-RPC invocation mechanism [FTW07].

The overall envelope consists of a *SOAP Body*, which encapsulates the actual payload in form of an XML-structured content and optional instructions for the SOAP processor within the *SOAP Header*. To be able to transfer non-desciptive data like binary files as part of an XML-based web service infrastructure, *SOAP with Attachments* (SwA) has been specified. In this case the SOAP message is combined with the MIME format to break down arbitrary data in successive data chunks, such as with E-Mail attachments [CJ02]. The wrapping and unwrapping of SOAP messages is performed through APIs as part of the web service environment. SOAP headers are often used to declare security-specific data to be processed and verified by intermediate *Security Gateways* before the payload is processed by the actual message receiver. In general, SOAP header data for *Intermediaries* plays a crucial role within the *Processing Model* of flexible web service infrastructures [FTW07]. The specification also considers a sophisticated message error handling through *SOAP Faults*. However, as with the processing of SwA, this functionality is not available for mobile web service environments until now.

3.4.4. SOA Analysis and Design

Platform-independent technologies like web services may only lead to successful long-term software solution if they are designed and implemented in the right way. It is therefore important to understand the anatomy of software achitectures and their relationship to the service concept. [KBS05] references to a *software* architecture from different perspectives, whereas in general it *'...is the set of significant decisions about the organization of a software system...'*, in more detail it *'...is a set of statements that describes software components and assigns the functionality of the system to these components. It describes the technical structure, constraints, and characteristics of the components and the interfaces between them. The architecture is the blueprint for the system and therefore the implicit high-level plan for its construction'*. The ability of distributed software systems to discover and to enable services across platforms is also refered to as the *Composite Computing Model* [CJ02]. This model is seen as the class to which a SOA belongs. A SOA typically refers to a business infrastructure based on a software architecture which contains *Application Frontends*, *Services*, a *Service Repository*, and a *Service Bus* as a system-specific communication infrastructure. The communication bus interconnects services and frontends in a technologic agnostic way and integrates pure technical services such as for logging, security, and transaction purposes. A service itself relies on a *Contract*[70], a service *Interface*, and a concrete *Implementation* of this interface in form of a *Service Stub* as well as on the actual business logic and the according data infrastructure. Figure 3.7 illustrates the relationship between the mentioned components.

These elementary SOA artifacts have different *Lifecycles*, since data and content may be a part of the system over decades, whereas application frontends are changing more rapidly. An application frontend might be a graphical web interface, a mobilized rich client, or even long-

[70] E.g. in form of an WSDL document.

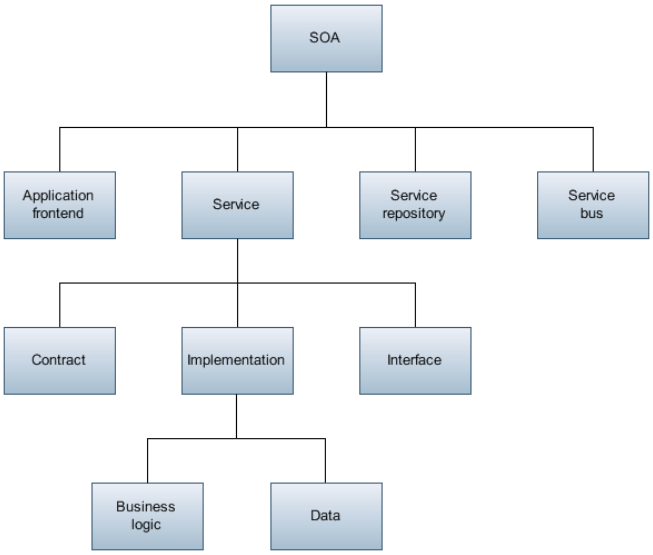


Figure 3.7.: *Artifacts of an SOA*

living scheduled processes without any user intervention. Frontends usually initiate business processes and receive their results. Business-oriented services also have different characteristics which may be seen as *Flexible Building Blocks* [KBS05] to manage complexity and to have a *Common Language* between the different stakeholdes of a SOA project. A block or component is situated within a specific SOA *Layer* depending on whether it is acting as *Basic*, *Intermediary*, *Process-centric*, or *Public Enterprise* service [KBS05].

3.4.5. SOA Implementation and Interface Issues

Since SOA-based systems have been implemented, interoperability problems still remained at a particular level. Many of that problems were solved by releasing and following according standards. System-specific and potentially long-lasting problems should however always be considered when designing and realizing large distributed systems. The cause of a time-consuming re-design or a plainly failed project is often found in a misusage of SOAP and WSDL themselves or an inadequate design. Basically there are *two* approaches to start implementing a SOA project: *Code-First* and *Contract-First* [FTW07].

3.4.5.1. Approaches

The most common approach for relatively simple projects is Code-First, whereby a WSDL generator automatically constructs the according service contract after the business logic is available. Furthermore, the resulting WSDL file is used to generate the service-related client code maybe through the same tool suite. This time-saving approch also protects the developer from manually creating rather complex service contracts. On the other side this strategy often causes interoperability problems as soon as the service is being accessed through a vast number of heterogenous systems. The reason lies in the occurence of different *XML Schema Definitions* (XSDs) with incompatible or unavailable data types [FTW07]. XML Schema defines a set of basic data types as well as possibilities for extending them to more complex application-specific

ones. The mapping between XML Schemas and platform- or language specific data types is therefore typically performed through standard binding frameworks with additional customized binding rules. In this case a developer is running into danger to support only data types which are currently available in the local development environment. As a consequence, service failures which have been emerging through non-standardized mappings rules when integrating with other systems are often hard to detect. There may also be problems with the WSDL contracts caused by deficiencies of WSDL generators. This and similar incompatibility problems are however considered through WSDL validators and the *WS-I Basic Profile* specification, which is well supported by appropriate frameworks.

As implied indirectly in the last paragraph, the interface-oriented[71] Contract-First approach is more suitable for large-scale projects. Since cooperative business systems are developed in a distributed manner and typically in stages, it is straightforward to develop against an existing service contract, designed and shared by multiple business partners. In this case a better control of cross-platform data type bindings and therefore an elimation of incompatibility problems can be widely ensured. Furthermore the client stub *as well as* the service code skeleton[72] can be generated automatically. As to Contract-First, it is crucial to validate the WSDL file before performing this step. To be more flexible in respect to subsequent changes, complex data types are usually defined as separate XML Schemas in *XSD Files*, which are combined to an overall XML Schema for describing the current message content [FTW07].

3.4.5.2. Java Web Service Support

Sun Microsystems offers several products for developing web services for the Java Enterprise Platform. They are categorized[73] in *Core Web Services*, *Enhanced Web Services*, *Secure Web Services*, *Legacy Web Services*, and services for *System Management* purposes.

As one of the core APIs for developing SOAP-based and RESTful web services, the *Java API for XML-based Web Services* (JAX-WS) takes off the role of JAX-RPC for this purpose. The *Java Architecture for XML Binding* (JAXB) package is an XML binding framework for associating Java objects with XML schemas and vice versa. As a bridge between Java web services and Microsoft technologies, the JAX-WS extension *Web Services Interoperability Technologies* (WSIT) ensures reliable cross-platform messaging and atomic transaction capabilities. Another framework for securing JAX-WS, JAX-RPC, and *SOAP with Attachments API for Java* (SAAJ) related services is *XML and Web Services Security* (XWSS). As representative of Sun's Legacy Web Services category, the *Java API for XML-Based RPC* (JAX-RPC) plays a crucial role for supporting SOAP 1.1 web services, based on current and former Java Platforms. This API is also useful if an `RPC/encoded` SOAP style is used. JAX-RPC can be used stand-alone or as part of an application server. According to [ibm06], there are currently three JAX-RPC-specific *Encoding Styles* for messages:

- `RPC/encoded`
  This is a straightforward approach and allows an easy dispatching of messages, however this approach decreases throughput performance.

[71] CORBA, DCE, and COM projects are typically based on this style.

[72] In cases where service code already exists, it is preferable to integrate the generated Java Beans directly into classes of this code. Some binding frameworks such as *JiBX* are supporting this strategy.

[73] http://java.sun.com/webservices/technologies/index.jsp

- `PRC/literal`
  This encoding style has the same properties as `RPC/encoded`, it is however compliant with the WS-I specification.

- `Document/literal`
  This style includes no type encoding info, whereby each XML validator can process the messages easily. However the operation name in the SOAP message is lost, with the consequence that a message dispatching might be impossible. This approach is also supported by the WS-I standards.

A mobile web service implementation typically uses the `Document/literal` encoding for mapping WSDL-based descriptions to actual *Java* objects, which is the safest option to interpret messages. Even if the *Java API for XML Processing* (JAXP) package is not directly WS-specific it is typically used to access XML parsers on the system.

Java Web service APIs are available since the release of the *JavaEE 1.4* platform specification. JavaEE-conform application servers such as the reference implementation *Sun Java System Application Server* (SJS AS)[74], its open-source version *Glass Fish*[75], *JBoss*[76], or *BEA WebLogic Server*[77] therefore have a built-in support for developing web services. Based on basic web service APIs, many tools and frameworks are available on the market with the aim to simplify the development process. A service developer's task is therefore also to handle a vast number of assistant software products, which sometimes makes an implementation appear prima facie more complex as it actually is. One product which tries to reconcile essential functionalities and tools is the powerful open-source SOAP engine *Apache Axis2*[78]. This engine is often integrated into *Apache Tomcat*[79] as a web service container.

3.4.6. Mobile Web Services

The objective of this section is to emphasize the meaning and the gaps of *Mobile Web Services* (MWSs) for B2B as well as for B2C application scenarios. As an important strategy for realizing largely distributed SOAs, MWSs are typically used to enable and to solve complex enterprise integration issues in the same way as their full-featured counterparts on wired hosts, but under a user-centric and mobile context.

3.4.6.1. Infrastructure

It is convenient for developers to let mobile end users act as *Web Service Consumers* (WSCs) to access and integrate with remote data and business logic residing on powerful network nodes offered through *Web Service Producers* (WSPs). They additionally shield the mobile from implementation details. Since web services are agnostic to any specific platform, the only

[74] http://www.sun.com/software/products/appsrvr/index.jsp
[75] https://glassfish.dev.java.net/
[76] http://labs.jboss.com/
[77] http://www.bea.com/
[78] http://ws.apache.org/axis2/
[79] http://tomcat.apache.org/

restriction in the mobile domain is limited hardware resources for handling document-centric protocols and dynamic network links.

Currently, mobile services are hosted by *Network Operators*[80] and *Internet Service Providers* (ISPs), which usually offer *Web Service Interfaces* (WSIs) to provide mobile application developers a suitable access to their business logic from heterogonous platforms. A WSP or even a *Web Service Broker* (WSB) is in fact not bound to the wired domain and may offer rich contextual information directly by mobile devices running a mini-web server, which let device owners act as service providers. In near future, a trend may go towards device-centric peer-to-peer web services, which might offer a *Collaboration-Oriented* [Cha06] communication pattern of high dynamically hosts. Furthermore, mobile WSBs might support social networks without the intervention of a network infrastructure provider. However, nowadays mobilized services are bound to service vendor infrastructures. Figure 3.8 demonstrates this situation by involving multiple domains.

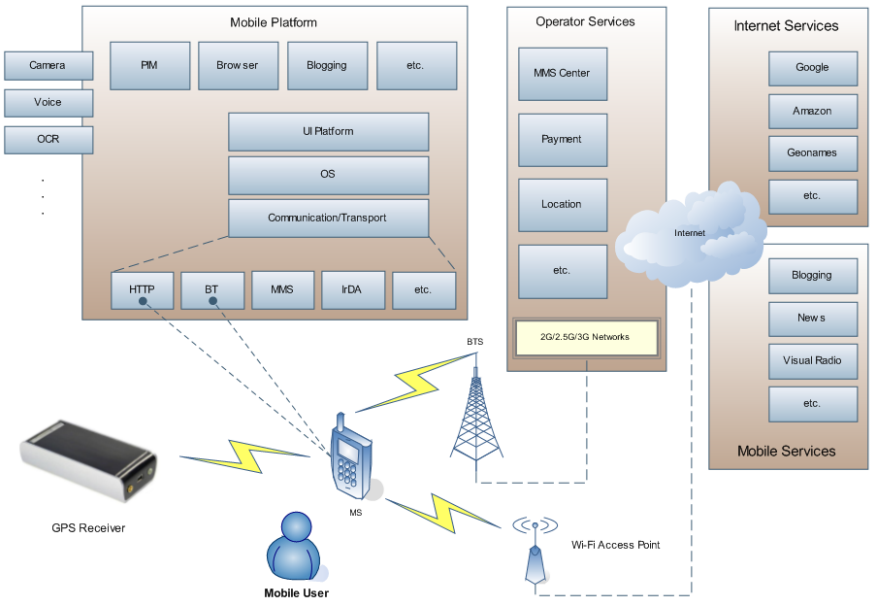


Figure 3.8.: *Mobilized Service Infrastructure including multiple Services from different Domains. A Bluetooth-connected GPS Receiver might be used to determine the geodetic Location of the User. The Location is a useful Parameter for Context-aware Services.*

3.4.6.2. Frameworks

Today the key mobile technology developers, which enable standardized frameworks for a mobilized service-oriented communication are mainly Nokia, Symbian, Microsoft, IBM, and Sun Microsystems, as well as a few open-source projects. According implementations are based on specifications from OMA, Liberty Alliance, JCP, Parlay, etc. (see appendix *Organizations* [A]). The *Nokia Mobile Web Service Framework* (NWSF) and the *Microsoft Mobile Web Service Framework* for *Windows Mobile* devices are currently the leading technologies in this domain [Cha06].

[80] E.g. Services for payment, SMS/MMS center access, user profile, etc. The operator may support a service access from a PC as well as from the mobile [Cha06].

The NWSF SDK is a middleware framework with a deep focus on security features and is situated between the developer and the networking facilities. It is a *WS-I Basic Profile*- and *WS-Security* conform implementation and provides access to SOAP-based messages through basic SOAP and XML processing libraries, and supports the developer with a sophisticated high-level *Service API* to manage service-related tasks more efficiently. The NWSF was developed for Series 80 devices, targeting both languages: C++ and Java. The S60 counterpart has identical features and is called *SOA for S60* and available as native platform component since the *third* edition[81]. Based on Symbian's Client-Server model, the SOA for S60 *Service Manager* is a robust server process, which is responsible for managing service connections as well as for storing service information and identities. The service manager also enables the registration of callback functions to perform *asynchronous* service requests by involving so called *Active Objects*.

The SOA for S60 architecture offers a pluggable interface[82] to be extended by specific frameworks, which are managing their own interactions, for instance through the wrapping and unwrapping of framework-specific message envelopes [HKI06]. A loosely coupled framework[83] is the implementation of the *Liberty Alliance ID-WSF* specification (see appendix *Liberty Alliance* [A.7]) for managing user authentications and their identities across several services. More precisely, a centralized *Identity Provider* (IDP) may be used by multiple services to fetch user credentials on demand[84] for performing a transparent authentication as soon as the client tries to access the service. This strategy only requires a single user intervention with the central IDP when starting a session with multiple secured and independent services, as in the case of a distributed booking system. This proven concept is manifested in *WS-Security* conform *Single-Sign-On* (SSO) systems like *Java Open Source SSO* (JOSSO), *CAS*, or the *Java System Access Manager* as part of the *Java Authentication and Authorization Services* (JAAS) module. The advantages of SSO are clear: simplified cross-domain authentication, convenient and more secure service interactions[85], and avoidance of fragmentation of authentication solutions for the enterprise. Mobile liberty-enabled WSCs therefore are of crucial importance when integrating with modular enterprise applications, where *'...authentication and other facilities are factored out of application services'* [HKI06]. Next to SSO features, SOA for S60 also supports *HTTP Basic Authentication*. Figure 3.9 illustrates a high-level architecture of the Nokia Web Service specification.

3.4.6.3. Service Environments

The JSR-172 implementation enables web services consumer applications on mobile Java platforms. The specification is also known as *Web Service APIs* (WSA) and supports the *WS-I Basic Profile* in the same way as the NWSF. WSA is based on SOAP 1.1, WSDL 1.1, XML 1.0, XML Schema, and HTTP 1.1 and therefore contains lightweight versions of JAX-RPC and JAXP [Cha06]. JAX-RPC enables *Stub Generation*, *Stub Instantiation*, and *Stub Operations* for including the web service interaction logic into the microedition code base as well as for configuring interaction properties. The client stub code is generated from WSDL files through according stand-alone tools or IDE web service features. The available XSD data types for

[81] However only for C++ applications until now [HKI06].
[82] Since Web Services are not the only service-oriented technologies, Nokia tries to meet future trends with this kind of architecture [HKI06].
[83] And the only one until now.
[84] Usually through a user-specific *Security Token* within the protocol header [Sam08].
[85] cf. *SOAP Message Security* encryption, *Replay Attack* protection, Identity-based *Discovery Services*, etc.

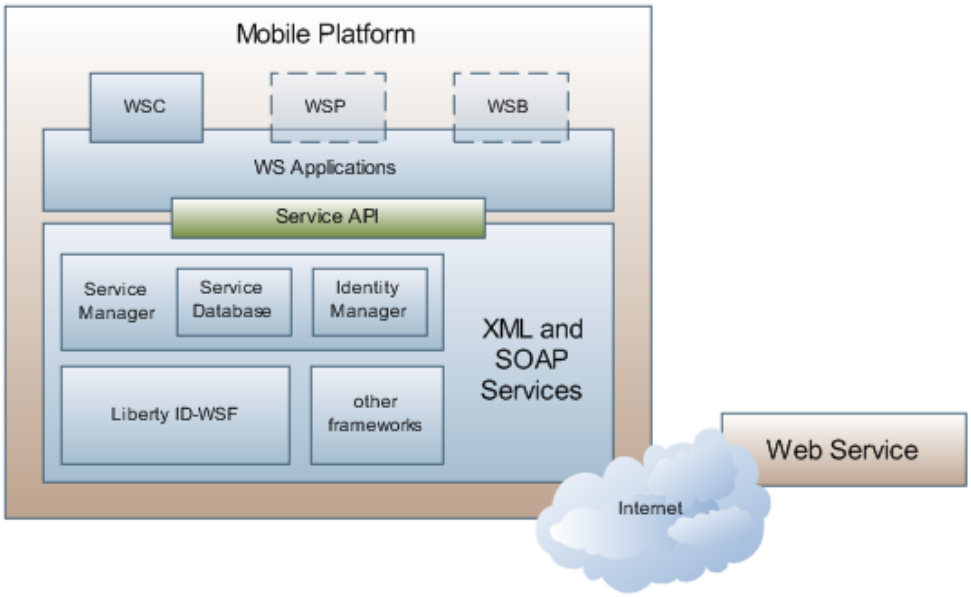


Figure 3.9.: *High-level Architecture of the Nokia Web Service Framework (NWSF)*

CLDC devices is limited to the basic ones and their mapping is adapted to the capabilities of UI platforms as well. This fact restricts the use of mobile web services, because server-side interfaces often involve XSDs which are not available in the mobile domain. The JAXP package references to the *SAX 2.0 API* and allows MIDlets to receive parse events. WSA implementations do not support *WS-Security*, SwA, WSPs, UDDI, SOAP message handlers, or an asynchronous communication model.

In cases where native mobile web service support in not available, third-party extensions like *kSOAP* in combination with *kXML* are frequently used. kXML was developed at the University of Dortmund and is an XML pull parser and writer. Both implementations have a very small footprint. Another related mobile extension is the ANSI C++ based *eSOAP* implementation. The aim of this work was to let embedded systems inter-operate with n-tiered application architectures. A further important tool for processing SOAP messages within embedded C and C++ based applications is *gSOAP* [Cha06].

3.4.6.4. Payload Communication Constraints

In contrast to XML technologies, suitable for rendering content directly for end user consumptions like XHTML-MP, WML, and SMIL, web service specific XML dialects are used for programmatic communication interfaces to remote resources. This textual and verbose communication formats have to be adapted for a mobile context by the environment. The properties of mobile communication infrastructures like cellular networks, Bluetooth, and messaging systems in different user interaction modes usually require a special attention in this relationship. The operator usually supports a compressed payload protocol to reach a more efficient exchange of plain XML structures over GPRS/UMTS networks and to save radio resources. One such a compression method is specified by the WAP Forum as *WAP Binary XML* (WBXML). This format preserves the element structure while encoding representative non-metadata XML tags as single bytes.

On mobile platforms the biggest deficiency of XML is its processing overhead and its lack to handle binary data via SwA. One alternative to transfer unstructured data such as images, is by encoding them as *Base64* strings, which however results in additional processing time and message enlargements by the factor 1.33 [Nok06]. In combination with WBXML, binary data might be transferred as so called *opaque* data [Nok06], which can be directly appended to the encoded XML structure without a pre-processing step. Consequently, the WBXML parser does not interpret this kind of attachments. The WBXML parsing capability is typically not a platform feature by default, but it can be made available through the integration of commercial or open-source solutions.

3.4.6.5. Service Availability

Since mobile users are usually charged by transmission volumes, rather than connection time, and a reliable responsiveness is important for the acceptance of a network-based application, it is preferable to optimize service connections. As already introduced in section *Payload Communication Strategies* [3.4.3] and *Payload Communication Constraints* [3.4.6.4], there are several approaches of how to speed-up remote server interactions a priori, by choosing from different communication models during the development phase.

Now the focus lies on a measurement of a service's availability within the mobile domain, by introducing *QoS* parameters. This strategy is worth to be discussed, since it generally enables an optimal end-to-end connection to arbitrary web-based services. Kee-Leong Tan and S.M.F.D. Syed Mustapha [TM06] propose an *MWS Availability Checking Model (MWS-ACM)* for integrating *Availability Metrics* into WSDL files, to give the mobile client feedback about the service status before a invocation is performed. Within the web service domain *QoS* refers to *Service Level Agreement (SLA)* to describe contracts and relationships between the service and its clients. According parameters might be *Availability*, *Response Time*, *Security*, and *Throughput*. With a concentration on obtaining the mobile service availability status, the according paper refers to *five* different groups of metrics:

- **Server Component Metrics**
  Connectivity to the *Web Server*, the *Application Server*, the *Database Server*, etc.
- **Server System Metrics**
  CPU Utilization, *Memory Consumption*, *Network Traffic*, etc.
- **Mobile Network Metrics**
  Response Time, *Latency*, *Throughput*, etc.
- **Message Size Metrics**
  WSDL-specific *Output Parameters* and their *Data Types*.
- **Device Metrics**
  Screen Size, *Media Processing Capabilities*, *System Memory*, etc.

In the work of Tan and Mustapha, the metric values are requested from mobile clients and computed on the server through the so called *Availability Checking Agent (ACA)* which is acting as a *Service Broker*. The agent finally delivers the service status data to the mobile client. The

Mobile Network Metrics are measured in the course of this initial data exchange. Since these metrics or based on error-prone wireless connections, they are the most unreliable in the chain. To enable an availability check with the MWS-ACM approach, the according XSD file with the availability elements has to be integrated into the WSDL file of the desired service. Next to this approach, there are similar works and efforts related to SLA, but often designed for the wired domain exclusively.

3.4.7. Java Service Platform

An important organization effecting the interoperability of Java based applications and services, is the *Open Services Gateway Initiative (OSGi)* Alliance, founded in 1999. The OSGi technology has been adapted by manufactures, service providers and developers and is an attempt to control the rising complexity when developing large distributed software systems by offering specifications and according reference implementions as well as test suites and certification programs. It is a contemporary non-proprietary technology with an improved time-to-market model by integrating pre-build and pre-tested subsystems into the business logic. The OSGi approach, originally aimed at residential Internet gateways for automatization of household applications, is now evolving in other market segments including the mobile and embedded sector. Nokia and other vendors plan to integrate the OSGi concept into their next generation smart phones. OSGi is also widely accepted by the Open Source Community as an advanced service-oriented, component-based *Universal Middleware* and is manifested in open source projects like *Apache Felix*, *Apache Derby* and *Eclipse Callisto* [OSG07a]. Moreover, according to [OSG07a]

> *'...the core OSGi technology is now increasingly prevalent in the Enterprise, and is also seen as the key component of the next generation Java Service Platform that enables the dynamic deployment of Web 2.0 Services and Mashups.'*

In detail, the middleware system is divided into containers, which are defining *Dynamic Software Component* interactions and a remote application lifecycle management, including over-the-network deploying and updating without service interruption (JVM restarting). Software components are Java libraries or according applications which are able to discover other components. The alliance has specified standard component interfaces, implemented by different vendors to suit the need of different markets, whereby OSGi provides a *Cross Platform and Cross Industry Service Platform* widely applicable from small devices to mainframes.

3.4.7.1. Domains

The OSGi service platform got popular within the *Enterprise Domain*. The reason may be the complexity of the *Java Enterprise Edition (JavaEE)* and its compatibility to other frameworks through mechanisms like *Inversion of Control (IoC)*[86] and *Dependency Injection*[87], which

[86] An OO paradigm which describes the working principle of frameworks: A function of an application is registered at a distinct library which calls this function at a later point in time a.k.a. *Hollywood Principle*. This means that the program does not have full execution control. This design pattern is manifested in *Listeners, Applets, Servlets, etc.*. IoC is also related to libraries which are working with *Dependency Injection*.

[87] A pattern for instantiation and initialization of objects while minimizing dependencies between them. Therefore this method can be seen as an abstraction of the *Factory Method Pattern*.

limits the software to a specific application environment. However OSGi does not address many JavaEE issues like APIs for persistence and messaging which are important within enterprise applications, whereas an OSGi enterprise expert group has been established in the end of 2006. Moreover an integration of OSGi technology into the popular and orthogonal *Spring* framework from *Spring Source '...will strengthen Spring's value as a basis for server infrastructure and offer benefits to users in the area of componentization, versioning and dynamic deployment.'* [OSG07b].

An interesting aspect is that the OSGi specification theoretically fulfills the requirements needed for a flexible, scalable, reusable and unified service platform for the *Mobile Domain*. That kind of platform is desired by mobile phone manufacturers, service operators and enterprises as well as by end-users, which are currently faced with unmanageable software customization and adaptions, potentially effecting more than half a billion[88] mobile Java platforms. The *JSR-232 Mobile Operational Management Submission* expert group's work is based on the OSGi concept to manage the life-cycle of mobile Java services, libraries and applications on-the-fly without any user intervention. This technology allows much more control over the enterprise and consumer platform domain and improves the support and the user experience in a device-independent way. This introduced *Client Middleware* will change the perspective within the mobile business market also by allowing numerous of new application fields [OSG07b].

3.4.7.2. Key Features

In the following, the working principles and the *Key Features* of the OSGi middleware are briefly discussed. First of all, the life-cycle controlling *Software Component Management* feature may install a corresponding `JAD/JAR` application bundle in the OSGi framework, which is being prepared to get ready for execution. The management component is now in the position to start, stop, update or remove distinct components through the framework API. This postulates the definition of additional `JAR` manifest headers and a reference to the framework API represented by the `BundleContext`-Object, within Java applications. Since the OSGi specification forms a small layer that allows multiple Java components to cooperate efficiently in a *single* JVM, the communication between running applications is as fast as possible while minimizing the memory footprint. Furthermore the service platform is designed to allow devices either run independently or under the control of the *Remote Component Management* feature, driven by a platform operator. In the second case an authorized software component has to map a distinct management protocol to an API call to get connected to the remote service. This concept provides the same interoperability as a standard protocol by additionally supporting protocols which are adapted to distinguish deployment scenarios. For instance, the *OMA Device Management (OMA DM)* is one of the OSGi supported protocols within the mobile environment [OSG07b].

3.4.7.3. Security

The OSGi middleware also simplifies the management of component configurations when developing software for this platform while considering *Security* aspects as well. The first level of defense is the security concept of the JVM itself, which is designed to restrict the

[88] millard in european terms

capabilities of an executing Java code by preventing for example dangerous buffer over-runs. The access modifiers of the Java Language represents the second level of defense, which allows a shared but shielded runtime environment for applications in a JVM. The final mandatory platform security feature lies within the OSGi framework that strictly separates bundles from each other by validating their *Service Permission* before being executed. In summary, the OSGi security architecture is quite advanced and sufficient to offer a reliable cooperation between OSGi platform applications [OSG07b].

3.4.7.4. Java Application Server Models

In contrast to other *Java Application Server Models*, such as MIDPv2 or JavaEE where applications are seen as private bundles which have to carry all their codes, OSGi *not* runs its code in a closed container. This means that the platform does not introduce an overhead when integrating with other applications. However, functionality is contributed to the environment and shared among other applications by downloading the distinct libraries resulting in much more slender code. Arising bundle dependencies and version control for classes as well as the loading of classes are solved by the OSGi middleware in a deterministic fashion. A closed MIDP environment has to provide a set of APIs to each single device to offer specific functionalities, instead of sharing them once within a collaborative environment. This environment requires a lightweight publish, find, and bind model for services *inside* the JVM, like manifested in the OSGi *Service Registry*. This loosly coupled component interaction supports a *Service Oriented Architecture*, whereas a registered OSGi service represents an object which is made available by one bundle and consumed by another one. These vendor-supported service objects have different granularity and might also be full featured services like a HTTP Server. There are several abstract standard OSGi services which can be integrated in specific solutions on demand (see appendix *Standard OSGi Services* [C.1]). More complex enterprise applications are often split up into sub-components for communication, database access, business logic, multiple user interfaces etc. During the OSGi deployment phase, components are then smoothly composed in a way to meet the requirements of a specific target environment, which also offers a simplified and flexible mobile application development [OSG07b].

3.4.7.5. Dynamic Platform Behaviour

The described *Dynamic Platform Behaviour* is one of the most interesting features, but also quite complex to maintain, since listener registries and listeners themselves must also become aware of the platform dynamics to be able to set their state according to bundle life-cycle changes. Based on a solid platform management, frequently component manipulations however do not influence the 24/7 availability of system components. Even servlets may be updated without forcing a restart of the underlying web server a.k.a. *Hot Update*. Furthermore, there is no strict policy for using the service platform, the maintainer has to assume the responsibility. OSGi Alliance members are in the position to certificate their implementations, whereas these applications are proven to run on service platforms of other vendors as well. In this case a service fee might be charged for the service consumer, but since different vendors are forced to compete against each other, the price is usually low and service quality high [OSG07b].

3.5. Location-based Infrastructures

Location-based infrastructures are highly distributed networks of services and components to provide the end-user with new kind of enhanced mobile applications. There are several ways to determine the location of mobile devices and its corresponding users. *Satellite-Based* systems, to name the most popular technology in this area, achieve a rarely high accuracy, whereas related receivers are compact and highly available at moderate prices. On high-end smart phones, a satellite service might be directly accessable through on-board hardware. Alternatively, telecom providers are able to offer a pure *Network-based* or an *Assisted-GPS* location service for end-users, using their 2G, 2.5G, or 3G country-wide wireless telecommunication infrastructure in combination with GPS stations. Another family of positioning technologies are *Short-range Network* positioning systems based on Wi-Fi, Bluetooth, RFID, IrDA, etc., but they are not in the focus of this master's thesis.

3.5.1. Location-based Services

Mobile data networks in combination with modern mobile devices and a seamless as well as a secure service/data access offer many new applications for an advanced end-user experience. The most attractive services with a great potential within the mobile domain are probably *Context-Aware* ones, related to the user's physical environment. In this sense, also the user's current location becomes a powerful aspect. According to [MS05] *'A typical location-based wireless service hosted by a service provider, analyzes the user's request, prepares a reply according to the user's location and sends the response.'* As described more detailed within the next sections, the positioning method can be somehow controlled by the service developer or even by the user, by involving different positioning systems. A standalone satellite-based service, for instance, does not depend on service operators. No privacy control has to be established as long as the retrieved location is not sent over the network. On the other side, location-unaware devices may have a fast access to a variety of operator-controlled LBS' with according security models. Sometimes, a combination of multiple positioning solutions is desired to ensure a maximal mobility and service quality.

The user's geodetic datum retrieved from a positioning service is suitable to be part of a further information service request originated from the mobile platform. The requested service might offer user friendly geographical addresses, weather forecasts, or real-time traffic situations with respect to the delivered datum[89], also without any intervention of an intermediate processing node. In such cases location-centric web services typically offer the desired data base.

3.5.1.1. Service Components

Typical location-based services are designed in a way to allow location-unaware handsets a widespreaded network-based application spectrum, which may be classified as following: *Emergency Services*[90], *Information Services*, *Tracking Services* and *Entertainment Services* [MS05].

[89] A LBS can also be refered to as a *Coordinate LBS*, whereby distinct coordinates are already arguments of queries.

[90] The US *E911* and *EU E112* specifications are location enhanced emergency services for mobile devices, which have to be supported by manufactures and operators. This capability is not exposed to developers [PD04].

From the operators perspective, a LBS is structured into *Service Channels*, *Service Context Layers*, and *Service Interfaces*. The location-related information is either *pushed* to, or *pulled* by the mobile platform, resulting in different service channel characteristics. Pulled-based LBS are much more propagated today, since they are easier to implement and allow the user to initiate a request on demand. The *Location Detection* context layer is the foremost layer, used to determine basic parameters like the longitude and latitude information of the mobile terminal. The *Location Transformation* layer is a context layer which is sometimes used to convert a location into a geographical landmark[91] to be represented within a cartographic interface. As a third layer, the *Location Tracking/Monitoring* layer provides location updates at periodical intervals. Operators may provide a LBS through different service interfaces/gateways: SMS, MMS, *Interactive Voice Response (IVR)*, Internet, or WAP.

3.5.1.2. Service Architecture

Including the presented components above, a LBS architecture contains a location access manager and a controller situated within a *Middle Layer*, surrounded by a *Device/User Layer*, a *Network Layer*, and an *Application Layer*. The *Device/User Layer* covers the user- and the corresponding service interfaces. The application layer includes the deployed mobile application which uses the LBS infrastructure.

The middle layer manages the user's privacy and profile information and offers client authentification mechanisms as well as interfaces to external engines, services and applications. The user may subscribe/unsubscribe services on demand, whereby full control over the location privacy is not guaranted. A user may have the permission to define a *Home Zone* and a remaining *Roam Zone*, which triggers the middle layer in cases where the user leaves or enters the cells corresponding to prior defined zones, resulting in different service fees [MS05].

The network layer represents all the location detection equipment, which is required next to the basic cellular network technology to determine the location of mobile terminals. The corresponding components belong to the operator's *Network Support Services* (NSS) infrastructure. Two important network layer components of this infrastructure are the *Gateway Mobile Location Center (GMLC)* and the *Service Mobile Location Center (SMLC)*, which provide location information of a requesting *Mobile Station* (MS) identified by its phone number. Before a query, which is received by a service interface, is being executed, application-specific *Criterias* like the maximal accuracy, the preferred location technique, and the service cost settings have to be retrieved. Afterwards, the GMLC redirects the query to the *Home Location Register* (HLR) to determine the user's registration data. This data is sent to the SMLC, which coordinates and assigns the resources for the determination of the user's position within the network. The actual query is performed by the *Mobile Switching Center* (MSC) and the corresponding *Base Station Subsystem* (BSS) through the *Location Measurement Unit (LMU)*, which is applying the positioning techniques described in section *Network-based and Hybrid Positioning Systems* [3.5.3]. The collected information is sent back to the SMLC, which finally returns the new location or the last successfully retrieved one (if the current positioning attempt failed) over the service gateway [Gut07]. Figure 3.10 illustrates this query process.

[91] *Map Servers* or *Geographic Information Systems (GIS)* are usually used to perform this task.

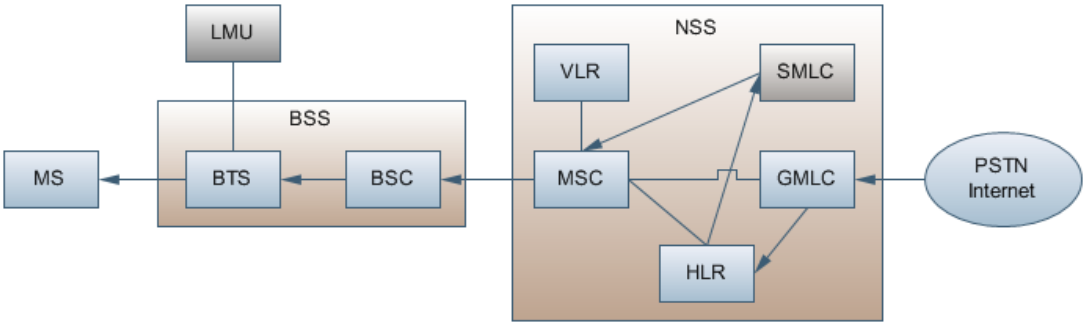


Figure 3.10.: *Location Determination within an Operator's Network Infrastructure*

When utilizing the privacy management infrastructure within the LBS middle layer, a service subscriber specifies whether/how the location information is controlled from other groups of subscribers or location clients. This configuration is performed by a SIM tool kit option, WAP, or SMS [MS05].

3.5.1.3. Protocols

Since a particular location-based application frequently uses foreign networks as well, a service *Standardization* is important to maintain the pervasive computing paradigm. The OMA-defined application-level *Mobile Location Protocol (MLP)* is for instance specified in a way to determine the location of the mobile terminal independent of the underlying technology. Such protocols have a great importance not only for the deployment of an area-wide location enhanced emergency system. To acquire location information on identificable Internet resources[92], the *Internet Engineering Task Force* and the *Open GIS Consortium (OGC)* developed geoprocessing specifications like the *Geographical Markup Language (GML)*, including location transformation and *Web Map Service (WMS)* definitions [MS05].

3.5.1.4. Programming Interface

A LBS offered through a network operator can for instance be accessed with the help of the JavaME-based *Location API*[93]. This programming interface is quite abstract. After the definition of the location criteria, the query is send to the corresponding positioning system, which triggers a location listener as soon as the result is available. The implementation of the *Location API* is usually aware of the available positioning methods by automatically searching services in the following order: Bluetooth connection to an external GPS service, internal GPS service, network-based service. In the next sections the working principle and the properties of the actual positioning technologies are described.

[92] E.g. The physical location of a specific server machine or a software.
[93] JSR 179

3.5.2. Satellite-based Positioning Systems

In the late '70s the NASA started to invest in cooperation with other US Departments billion of dollars for establishing a strategically important global object localization and navigation infrastructure based on 24 satellites on 6 different orbits (about 19000 km over the earth's surface). In this height, a satellite circumnavigates the world in approximate 12 hours. In 1995 all the satellites reached their final position. This *Global Positioning System (GPS)*, originally known as *Navigation Satellite Timing and Ranging (NAVSTAR)*[94], consists of three main segments: A *Space Segment*, a *Control Segment*, and a *User Segement* [GAR00]. The space segment contains 21 active and three spare satellites, which are distributed around the earth in a way to have a straight view to at least *four* satellites from every position on the earth's surface. The control segment consists of five distributed terrestrial base stations with one manned master control station to track and update the satellite system. The user segment consists of the potential user base like military, hikers, pilots, hunters, etc. and their corresponding receivers to be able to navigate through a specific terrain or to annotate data with location information for further processing.

3.5.2.1. Positioning Technique

The civilian GPS receivers receive low power satellite radio signals in the scale of 20 to 50 watt with a frequency of 1575,42 MHz by using the *Ultra High Frequency (UHF)* band. In contrast to a 100,000 watt powered local FM radio station on earth, GPS signals are usually not able to run through massive buildings or even not through a thick cloud ceiling. A *Line of Sight* to the satellites should be ensured while using the service, which usually depends on weather conditions. The receiver has to detect the space positions of at least *four* satellites and the distance to them, to be able to determine its own position on the earth's surface. The accuracy is proportional to the number of visible satellites. The transmitted and *coded* GPS information contains the satellite positions a.k.a *Almanac* data, which also allow to determine the signal travel time a.k.a *Time of Arrival* to the receiver, used to figure out the *Distance* between the communicating nodes [GAR00].

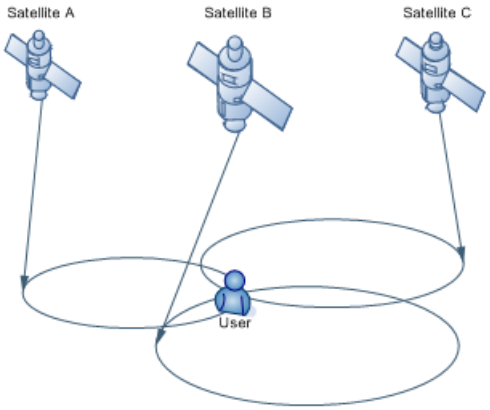
The *Almanac* data of satellites are stored and updated periodically within the receiver's memory. An update is made by the receipt of the *Ephemeris* data which contains the corrected satellite positions initiated by the master control station when a satellite travels slightly out of orbit, which happens every 4 to 6 hours. A GPS receiver is considered as 'warm' when it has stored the latest *Ephemeris* data of viewable satellites, otherwise it is said to be 'cold'. Therefore a GPS receiver start-up takes some time to aquire satellites for performing a position calculation. This so called *Time To First Fix (TTFF)*[95] [Sur07] lies in the scale of several seconds up to more than one minute, depending on the physical environment and the used infrastructure (see section *Assisted GPS* [3.5.3.2]). A small TTFF or satellite lock-on value is considered as an important aspect when acquire a GPS unit, especially when a reliable tracking of high dynamic objects is desired. Stable satellite locks may be maintained at all times through the usage of multi-channel GPS units, whereas multiple circuits are devoted to a particular GPS signal[96].

[94] Nearly parallel the deployment of the NAVSTAR satellites the former Sowjet Union started the *Globalnaya Navigationnaya Sputnikovaya Sistema (GLONASS)* project, which was given up after the collapse of the Sowjet Union in 1991. However Russia has re-adopted the project and plans to reach its final stage in 2010. Currently, also a European Global Positioning System called *Galileo* is under construction.

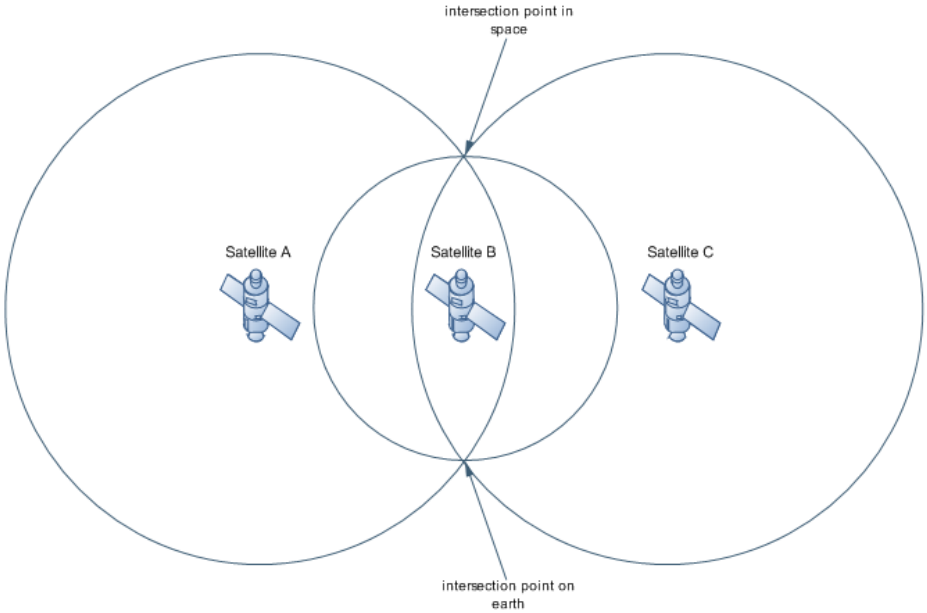
[95] The term *Fix* is associated with the actual position data.

[96] This approach reduces the TTFF value and improves the ability to receive signals even under difficult conditions

Within the receiver, the radio wave's velocity c is being multiplied by the signal travel time Δt resulting in the distance d between sender and receiver. Due to the earth's atmosphere, the signal velocity lies slightly under the velocity of light. This circumstance is considered through an ionospheric delay model within the final formular. The receiver is responsible of calculating the distance with the help of a received, satellite-specific *Pseudo-Random Code (PRC)*[97], which is compared with a similar receiver-specific PRC [GAR00]. The aim of this operation is to shift both codes until they match, whereas this shift Δt is rated to the signal's travel time between the nodes. The calculated distance d=Δdt*c to three satellites is theoretically sufficient to determine the exact position on the earth's surface like illustrated in figure 3.11.



(a) *The Intersection Point of three Sphere Surfaces results in a precise Position on the Earth's Surface (The Spheres are not shown to the Clarity)*



(b) *A further Intersection Point lies in the Space and has no Relevance*

Figure 3.11.: *Principle of the Positioning Determination with Satellites*

like in urban settings.

[97] Each receiver knows the PRC of each satellite, whereas it has the ability to filter specific PRCs from an received overlayed signal. The signal is refered to as pseudo random, since it looks like a noise signal.

Due to its inefficiency, a GPS receiver's internal clock is not synchronized to a atomic time like in the case with satellite clocks (*GPS System Time*). The distance measurement therefore contains an error[98] resulting in a *Pseudo-Range Data*, which has to be corrected by using the fixes of at least *four*[99] satellites [GAR00].

With the civilian GPS service, further errors like multi-path signals caused through object reflexion especially within urban areas and therefore a corruption of the satellite distance value are common problems. A suboptimal satellite geometry[100] and especially the so called *Selective Availablity (SA)*[101] are further sources for an unprecise location evaluation within civil applications. Due to economic reasons, the SA restriction is however inactive since 2000, which improves the positioning accuracy from 50-200 meters to 6-12 meters[102], which offered many new applications like consumer navigation systems. Independent of the removal of this restriction the GPS service is divided in two categories, refered to as *Standard Positioning Service (SPS)* for civilian applications and *Precise Positioning Service (PPS)*[103] for US military applications with an accuracy up to a few centimeters [Häß01].

3.5.2.2. Overlayed Systems

To overcome some of the errors mentioned above which are mostly related to the SPS service quality, the *Differential GPS (DGPS)* represents a suiteable approach. With DGPS, a terrestrial reference station at a known location is continously measuring its current GPS location and is therefore able to send[104] a calculated GPS signal error for tracked satellites in form of a DGPS correction message to a moving object related to the station. The DGPS receiver is therefore able to correct its own location by considering a known GPS signal error value for distinct satellites, resulting in an accuracy of 1 to 5 meters [GAR00]. A similar approach as DGPS is the *Wide Area Augmented System (WAAS)*, which uses a geostationary satellite for transmitting the correction data [Rot02].

3.5.2.3. NMEA Protocol

In 1983, the *National Marine Electronics Association (NMEA)* defined a standard protocol for exchanging data between an GPS receiver and an external processing unit. This protocol was

[98] A time difference of only 1μs between the GPS system time and the receiver's internal time causes a location error of about 300 meters [Rot02].

[99] The receiver's clock can not be synchronized with satellite clocks to calculate an exact distance, since this synchronization signals would have to travel faster than the speed of light. A further problem, which is also related to the theoretical physics domain is that a clock situated in the interstellar space is influenced by the gravitation of the universe and therefore not working under the same conditions as a clock on the earth. To solve this problem, a forth satellite must be used to calculate the user's exact position. A description of how this method works can be found in [Rot02]

[100] A satellite geometry is seen as ideal if the nodes are located at wide angles relative to each other. A minimum elevation angle limit is often built into GPS units to avoid using satellites close the horizon, whereby the signal would have to travel through more atmosphere resulting in more unprecise values [Sur07].

[101] An introduction of an artificial location data imprecision for civil applications to avoid a GPS accuracy which is nearly as high as for military applications.

[102] Official location-technology specific accuracies have always a probability of 95%.

[103] With PPS another PRC code with a higher frequency is modulated resulting in much more precise satellite distance calculation. The PPS signal is encryped and only accessable by the US military and the NATO [Rot02].

[104] The correction message is transmitted in real-time e.g. over FM radio frequencies or satellites.

named NMEA 0183 and is nowadays the standard protocol for handling raw GPS location information using an asynchronous RS-232 compatible serial interface with a default baud rate of 4800[105]. NMEA 0183 devices are designed to be *Talkers* or *Listeners* or both. The GPS talker information is provided in NMEA sentences built upon ASCII characters only, whereas a listener is querying the talker for a particular sentence of interest. Often only a talker is used, which is constantly sending NMEA sentences over its opened interface. Each sentence is starting with a `$` character, followed by a 2-digit talker ID, a 3-digit sentence ID, the actual payload, a sometimes optional checksum starting with a `*` character and an obligatory carrage return/line feed `<CR><LF>`, resulting in a maximal length of 82 characters [NME07]. The NMEA 0183 standard also allows the definition of proprietary vendor-specific sentences starting with a `$P` and a 3-digit vendor ID as well as an NMEA conform data representation. This also allows a specification of the device's baud rate and the underlying geodetic format [Häß01]. The *Recommended Minimum GPS/Transit Data (RMC)* sentence defines the minimum recommended GPS information delivered by each GPS unit, containing at least the longitude and latitude information of the current position (see section *Positioning Models* [3.5.4]). In the following table the fields of a valid RMC sentence (as received from a GPS unit) are analysed [Häß01].

```
$GPRMC,143029,A,4844.6196,N,12311.12,W,000.0,054.7,230500,000.2,W*7C
```

Field Value	Description
GP	Talker Identifier (Global Positioning System Device)
RMC	Sentence Identifier
143029	Timestamp of Sentence Creation (14:30:29 GMT)
A	Receiver Warning (OK - Valid Data)
4844.6196,N	Latitude and Hemisphere Direction (48 deg. 44.6196 min North)
12311.12,W	Longitude and Hemisphere Direction (123 deg. 11.12 min West)
000.0	Speed over Ground (0.0 knots)
054.7	True Course
230500	Date of Last Fix (23.05.2000)
000.2,W	Magnetic Variation (0.2 deg West)
7C	Hexadecimal checksum value of NMEA Sentence

Table 3.1.: *NMEA 0183 RMC Sentence Example - List of Sentence Fields and their Description*

3.5.3. Network-based and Hybrid Positioning Systems

Next to the incorporation of the GPS service, fixing an user's location is also possible through the already established cellular networks of telecom providers as indicated in section *Location-based Infrastructures* [3.5]. The major advantages with this approach are that the additional investment in network components is quite low and the according location techniques are working as long as the mobile device is able to establish a connection to the surrounding base station of the provider's network. A further advantage is, that with network-based systems no additional hardware is necessary as part of the user's equipment. On the other side, the LBS is usually not free of charge. Furthermore, this outdoor/indoor approach only has an average accuracy of several kilometers in rural areas where the GSM cell sizes may have

[105] The NMEA 0183-HS standard as part of the NMEA 0183 v3.0 specification uses a default baud rate of 38400 [NME07].

35 km, and up to 50 meters in urban areas with cell sizes of approximately 100 m. The quality of this less accurate location service in contrast to satellite-based ones, obviously heavily depends on the network/human population dense. Figure 3.12 provides an overview of available network infrastructure-based positioning techniques in contrast to GPS enabled ones.

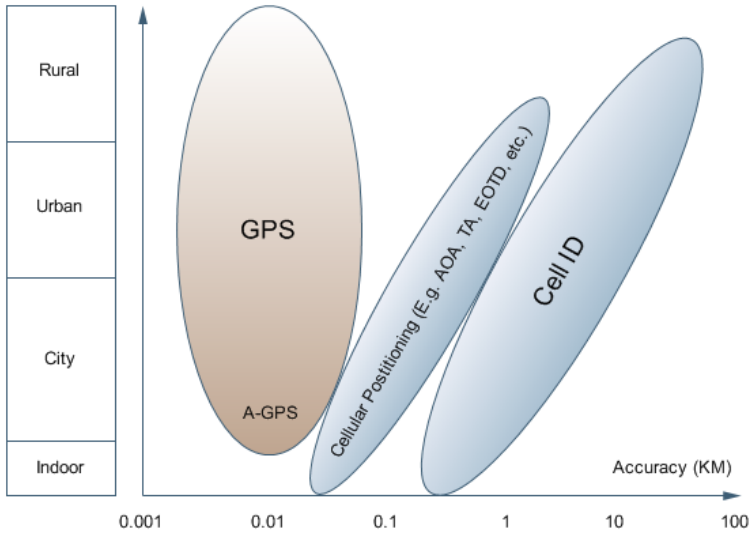


Figure 3.12.: *Positioning Techniques for the Mobile End-User in Dependency of the received Accuracy and the Environment of the Mobile Device*

3.5.3.1. Positioning Techniques

Since a GSM network is always aware of the communicating *Mobile Stations* (MS) within a distinct cell, a coarse positioning of a MS is a-priori given. The resolution is in most cases however not as exact as desired by corresponding applications. For this reason an additional infrastructure is needed to constrain the position of a MS on a maintainable level. This network-based positioning methods are often combined. Their names already expose information on how the technology works: *Cell of Origin (COO)* or *Cell ID*, *Angel of Arrival (AOA)*, *Time Advance (TA)*, *Uplink Time of Arrival (UL-TOA)* and *Estimated Observed Time Difference (EOTD)*, all with various advantages and drawbacks in respect of accuracy and complexity. The COO, which only uses the position of the next base station as final MS position is very unprecise and therefore often improved by dividing a cell in *Cell Sectors (CS)* established through an BSS with multiple directional antennas working at different frequencies. A further improvement of the COO or COO CS approach to restrict the possible user location within a cell is through a measurement of the *Received Signal Strength (RSS)*. The AOA approach makes only sense if the base stations use expensive antennas with direction characteristics, whereas they are able to calculate the position of the MS through a measurement of the incoming signal direction [Gut07]. Mobile stations and base stations are using exact time slices to communicate with each other, whereby this approach has to consider the signal runtime between the nodes as well. By default, the TA method is used to initiate a signal transmission from the MS at time in advance which is proportional to the distance from the BSS. This information can be used to estimate the position of the MS within a cell in steps of about 550 meters. The UL-TOA method can be used if the MS is in the coverage of four base stations. A signal runtime measurement and an evaluation

similar to the GPS approach allows a position determination with an accuracy of 50 to 150 meters [Rot02]. The EOTD method is similar to the UL-TOA one, with the difference that the position of the MS is calculated by the device itself, after the necessary information has been received from the operator's LMU. The evaluation parameters are once again the signal runtime differences between signals emitted from the base stations [Gut07].

3.5.3.2. Assisted GPS

Nowadays, the support for network-based LBS is spare[106], but it may have a great protential especially as a hybrid Network-/Satellite-based positioning approach also known as *Assisted GPS (A-GPS)*[107]. A-GPS is for instance very helpful in urban canyons where the satellite acquisition time may take several minutes or if a lock-on is impossible. Assisting network components bypasses the cold start phase of GPS units by sending pre-determined *Ephemeris* data (see section *Satellite-Based Positioning Systems* [3.5.2]) through a wireless network link to the requesting units. The *Ephemeris* data is determined through a *Reference GPS Receiver* which is a sub-component of the SMLC. As soon as the MS receives the assistance information from the network, the position calculation is either performed by the MS itself in combination with the viewable satellites, or by the NSS infrastructure. In the second case the relevant satellites are determined through the known location of the BTS to which the MS is locked-on[108]. Afterwards these satellites are used to calculate the position of the MS [Tay05]. The NSS approach futhermore saves resources on mobile units. The accuracy for A-GPS lies between 5 and 30 meters and the TTFF value is usually *less than ten seconds* [MS05]. A-GPS also supports a better roaming and does not need expensive network installations like LMUs and associated equipment. Figure 3.13 shows the infrastructure of A-GPS.

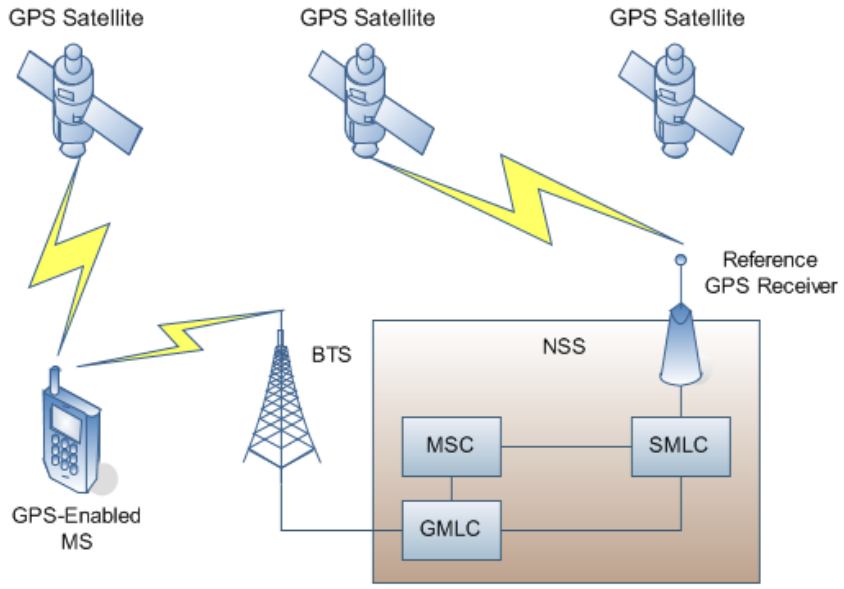


Figure 3.13.: *Schematic Infrastructure of an Assisted GPS System*

[106] For instance, in Germany some providers like T-Mobile, D2 Vodafone, E-Plus, and O2 offer location-based services over an WAP portal or an SMS gateway [MS05]

[107] In Japan this encouraging technique is already deployed with success.

[108] As approximation to the MS location

3.5.4. Positioning Models

There are several mathematical models for a description of a specific position on the earth's surface, divided into two subsets: *geometric* and *symbolic* models. Geometric models contain further sub classes: *Earth-Centered Earth-Fixed (ECEF) XYZ Coordinate Systems*, *Conformal Coordinate Systems*, and *Geographic Coordinate Systems*. All geometric models use coordinates within the model's reference system to point on specific positions, which are in contrast to symbolic models suitable to be processed by computers. A geographic model uses a reference ellipsoid as a model for the earth's volume. This *geodetic*[109] shape which may be seen as the base for a *Geoid*[110] is mainly defined through its *Equatorial Radius* (semi-major axis) and *Polar Radius* (semi-minor axis), sometimes together with the earth's *Flattening* at the poles. The *Geographic Model* is the most important positioning model and the *WGS84 Datum* the most important map datum format, also used by the GPS infrastructure. However a GPS receiver may handle a dozen of different positioning or grid formats as well as more than hundred map datum formats [PD04].

3.5.5. WGS84 Datum

The *World Geodetic System of 1984 (WGS84)* geoid is currently the most widely adoped earth reference system since it is precise and user friendly. The classical WGS84 datum as a mathematical representation of a point on the earth's surface, consists of a *Longitude*, a *Latitude*, and an *Altitude* value. The longitude value is refered to the plane which totally covers the equator, and the latitude value is refered to the plane which totally covers the *Main Meridian* going through *Greenwitch* in UK [Häß01]. This principle is illustrated in figure 3.14.

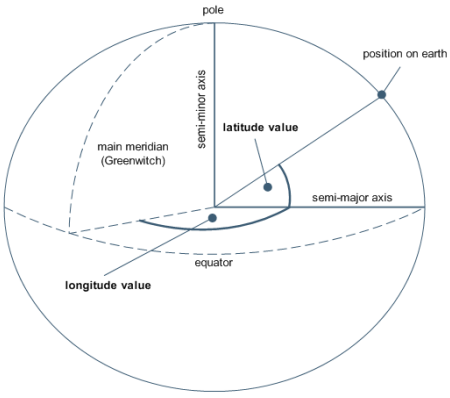


Figure 3.14.: *Definition of the Longitude and Latitude for the WSG84 Earth Reference System*

The WGS84 values are often represented as the deviation in degrees, minutes, and seconds from this reference planes together with the hemisphere direction if the denoted values are not signed[111]. The WGS84 ECEF format, which is not as understandable to humans as the classical WGS84 format, is however the most popular one when it comes to computer communication and processing of map datum information [PD04].

[109] *Geodesy* refers to as the science of measuring and mapping the earth's surface

[110] An abstracted model for the shape of the earth considering an earth's abstract model gravitation.

[111] 1 degree of longitude at the equator is roughly 111.3 km, whereas at 60 degree of latitude, 1 degree of longitude is not more than 55.8 km [PD04].

4. Requirements

"A journey of a thousand miles must begin with a single step."

— Chinese proverb

The chapters *Introduction* [1] and *Background* [2] provided an early position on the problem of realizing a distributed location-based application for the aggregation and management of mobile-generated media content. Such a functionality is going to be manifested as the core system of a *Mobile Blogging Application* called *TMBlog*. The implementation also aims to verify the adequacy of the corresponding solution for distinct application scenarios. Additionally, this mainly user-centric blogging tool focuses on privacy and usability aspects, which usually have a great impact on an acceptance of the system through device owners. The technologies and concepts which have been discussed in chapter *Analysis of Related Technologies* [3] are in a great measure the foundation for building this prototype and to understand its runtime environment as well as design decisions. This chapter describes *Tourism-related Studies*, possible *Application Scenarios*, the *Application Infrastructure*, derived detailed *Functional Requirements* for the prototype and its possible extensions, as well as its *Communication Models*.

4.1. Tourism-related Studies and Deployment

Tourism and LBS' are often seen as an ideal mix, because travellers are often confronted with unfamiliar places and a recording of locations for the *'generation of an authentic travel experience'* might be desired. [Leu05] predicts LBS' as a key-defining factor with a high potential application field in the mobile industry. The University of Auckland accomplished a study on the perceived value of LBS from a customer-centric perspective in the New Zealand tourism. The results of this study lead to a set of requirements for according mobile solutions in terms of user preferences and perceptions. In general, a successful introduction of such services usually depends on the usefulness, performance and costs of according solutions. In areas where the tourism makes up a great part of the country's *Gross Domestic Product (GDP)*, an efficient and novel deployment of tourist-related services might have a great impact on the visitor's immersive travel experience and behaviour. Unlike as in Austria and in other populous areas, different issues such as a fragmented coverage of the mobile Internet in isolated terrains like deserts, rain forests, mountain chains, etc. must be considered when designing mobile applications for world-wide travellers. The aimed solution in this master's thesis is not tightly integrated into large-scale provider networks or into a country's tourism service infrastructure in general. However, consumer-centric studies for such exhausting and possible long-lasting IT systems are also valid in small dimensions. The requirement analysis for the aimed LBS-based blogging prototype therefore heavily depends on experiences with such tourism services and according studies.

4.1.1. Perceived Value of LBS

Clara Lueng carried out a research in two phases including in-depth interviews with expert informants in the mobile industry and technology-interested international travellers [Leu05]. This evaluation process formed *a much fuller picture* of LBS issues in the tourism sector. The expert informants defined necessary requirements for mobile LBS solutions as follows: *Usefulness*, *Ease of Use*, *Security & Privacy*, *Personalisation*, and *Interoperability*. The related consumer study with 31 travellers in New Zealand gave an insight into the technology-experienced mobile user's preferences, doubts and perceived risks when using such kind of services [Leu05].

More exactly, the consumer interviews manifested different important values for travellers when using LBS', grouped into *Contextual Value*, *Functional Value*, *Social Value*, *Emotional Value & Emotional Risk*, and *Linking Value*. These values result from a *Utilitarian Thinking* on the one side and a *Hedonic Thinking* on the other side. Like expected by the researchers, the *Contextual Value* with its key factors *Mobility* and *Context-Awareness* has the greatest importance for a service consumer. Furthermore, *'service costs appear to be the most important functional value, while ease of use, reliability, convergence and global standards, are assumed to be inherent in the service, otherwise they will simply reject and avoid using the service altogether'*. The *Social Value* is related to interpersonal interactions, such as keeping in touch with home and other travellers for sharing experience and to enable a *local word-of-mouth communication*. This strategy leads to a *Linking Value*, which fits well to the concepts of postmodern individuals *'who are highly autonomous and mobile in a spatial and social sense.'* [Leu05]. This technology-based communication forms a strong sense of community among travellers and lets them stay connected regardless of their physical location. *Emotional Value & Emotional Risk* are crutial factors for the acceptance of services. The utilization of LBS' supports the *Self-Assurance* through the establishment of advanced emergency services. A single traveller typically perceives a more secure or safe feeling during a journey if someone is able to recognize abnormal situations as well as his last location in case of an serious accident. However, privacy invasions and the possibility of personal data misuse are further strong emotions, which might reject a usage of LBS' through the mobile user. The trustfulness of a service is typically higher, if the user has full control over service parameters and if the service offers expected information. Furthermore, *'fear of biased information is a perceived risk of using location-based tourism information services,...biased by other consumers with opinions...or via sabotage by firms.'* [Leu05]. In this sense, a printed travelling guide is still a preferred assistant for travellers, even if the information is not up-to-date. Finally one has to consider that each medium has its own qualities. Technology-interested travellers might however tend to utilize reliable and well integrated IT services as well, which offers much benefit also in terms of flexible communication models.

4.1.2. Application Scenarios

In the following a specific scenario is described to get more immersed into a traveller's situation when using the *TMBlog* solution: A person might be on a long *Journey* possibly on another continent. In such a case the application, installed on his smart phone, might be used to create location-based blogs to get aware of *Distances* covered by plane or other means of transportation. Furthermore the traveller would be able to determine the province and the state he is staying at the moment or he has previously visited, by exploring his trace on the phone. Additionally the mobile application might be used to *communicate* with relatives or friends at home by posting created multimedia blogs into a web environment. Consequently it is a good

way to be informed about someone's experience and adventures without suffering high costs for phone calls or time management. Extensions might allow further services such as weather forecasts for the current region.

A further already mentioned application field for the *TMBlog* approach besides tourism is the *Research and Statistics* sector: The application might be a handy tool for *Documentation* purposes in scientific settings. A biologist might for instance track the occurance of distinct plants or flowers within specific areas, by taking photos and by optionally adding some annotations with a mobile handheld device. The collected location-based data might be transfered to an institute's server immediately, to be automatically prepared for a convenient cartographic exploration within a web environment. The offered spatial circulation and according elevation levels where distinct species are situated might have relevance for the biologist's studies. The transfered geotagged data might also be stored and pre-processed as application-independent XML-representations, suitable for a transformation and an integration into an arbitrary context.

4.2. Application Infrastructure

On principle, the *TMBlog* application is distributed over three domains: *Mobile Client Domain*, *Server and Backend Systems Domain*, and *Web Client Domain*. The connectivity between the mobile and the static components typically not relies on a particular network.

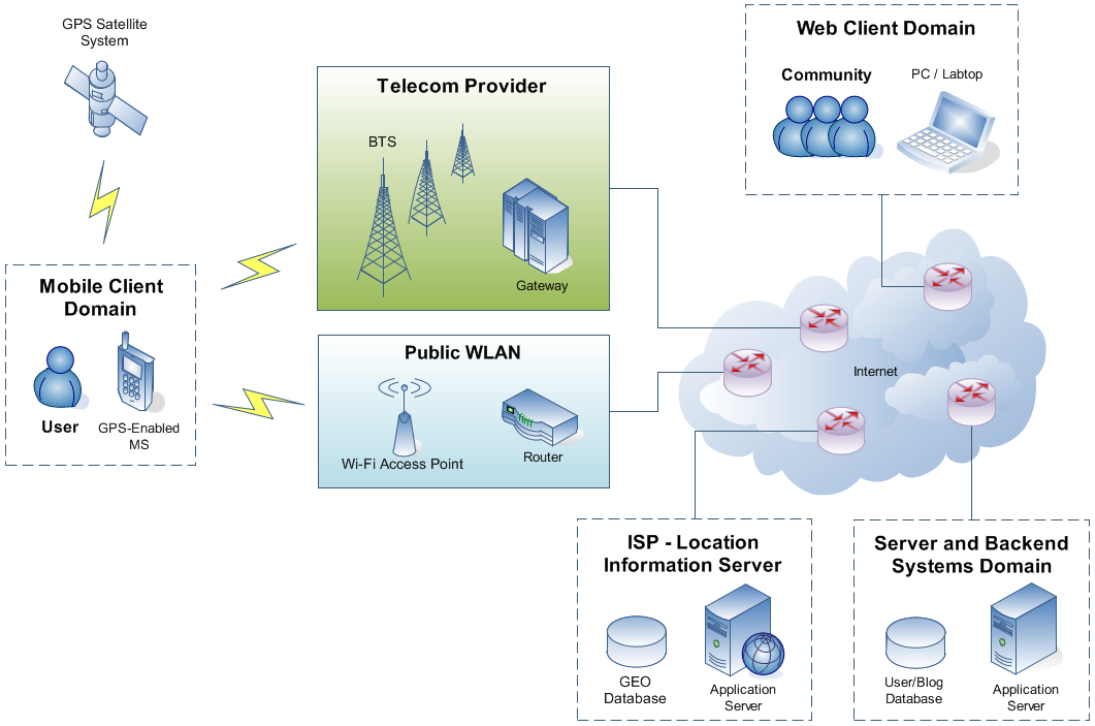


Figure 4.1.: *High-level Overview of the Blogging Application's Infrastructure. Multiple transfer Channels provide a flexible Interaction with the Server and Backend Systems, and the Community respectively.*

Country-wide provider networks support the highest possible mobility and flexibility to communicate with remote nodes. However the availability of further wireless technologies like WLAN and Bluetooth through modern handsets also enables alternative and charge-free communication channels. A utilization of the introduced GPS-enabled handsets in the last years increases the application's usability, since no additional hardware would be necessary. Figure 4.1 illustrates a high level overview of the application's infrastructure and the mentioned communication technologies.

WLAN access points are discovered by WLAN-compliant devices through a native platform functionality and can be chosen in the same way as data-centric access point providers by end users as soon as the application initiates HTTP connectivity. A Bluetooth transfer channel however requires a special focus, since it is not suitable for accessing Internet-based network nodes by default. Such an optional short-range channel might be part of the application's communication logic. As a mainly social application it is desired to reach a huge user base and therefore as many mobile devices as possible. The wide-reaching availability of KVM-enabled devices theoretically[1] fulfills this fundamental requirement.

[1] As mentioned in section *Mobile Software Development* [3.2], errorneous implementations through vendors are common.

4.3. Functional Requirements

In the following sections, the application domains as they have been indicated in figure 4.1, are going to be analysed in detail, to be able to launch the *Prototype Design* [5] and the *Prototype Implementation* [6] phases. The *Functionality* mainly focus on user-centric application aspects and on high-level system components as well as on first ideas of how to realize them. The actual decissions in this development stage are mainly based on consumer studies in the tourism sector like mentioned in section *Tourism-related Studies and Deployment* [4.1].

4.3.1. Mobile Client Domain

The according MIDlet is divided into five major domains: *System*, *Security*, *Content*, *Location*, and *Communication*. The following figures visualize all desired user interactions with this domain in form of *UML User Case Diagrams*, which are analysed and discussed in the next sections. These features should be finally implemented as part of the prototype. In figure 4.2 it is assumed that blogs are already stored on the device.

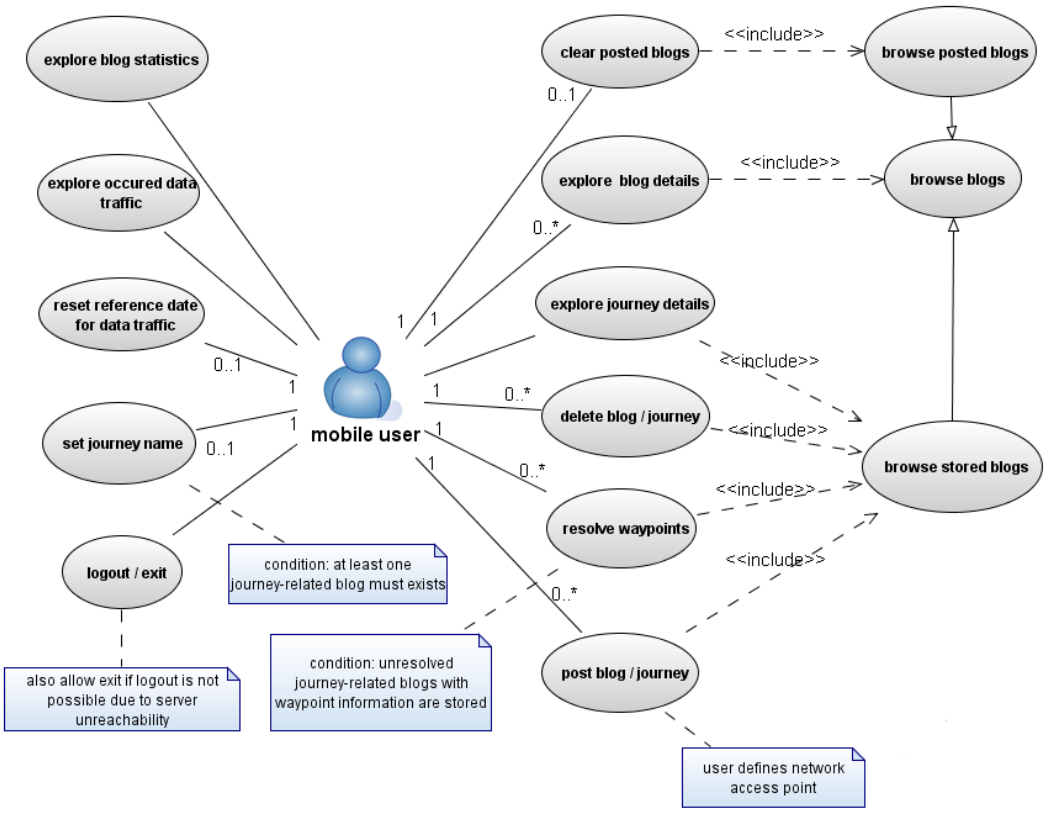


Figure 4.2.: *UML User Case Diagram for User Interactions in the Mobile Client Domain: Application and Content Management*

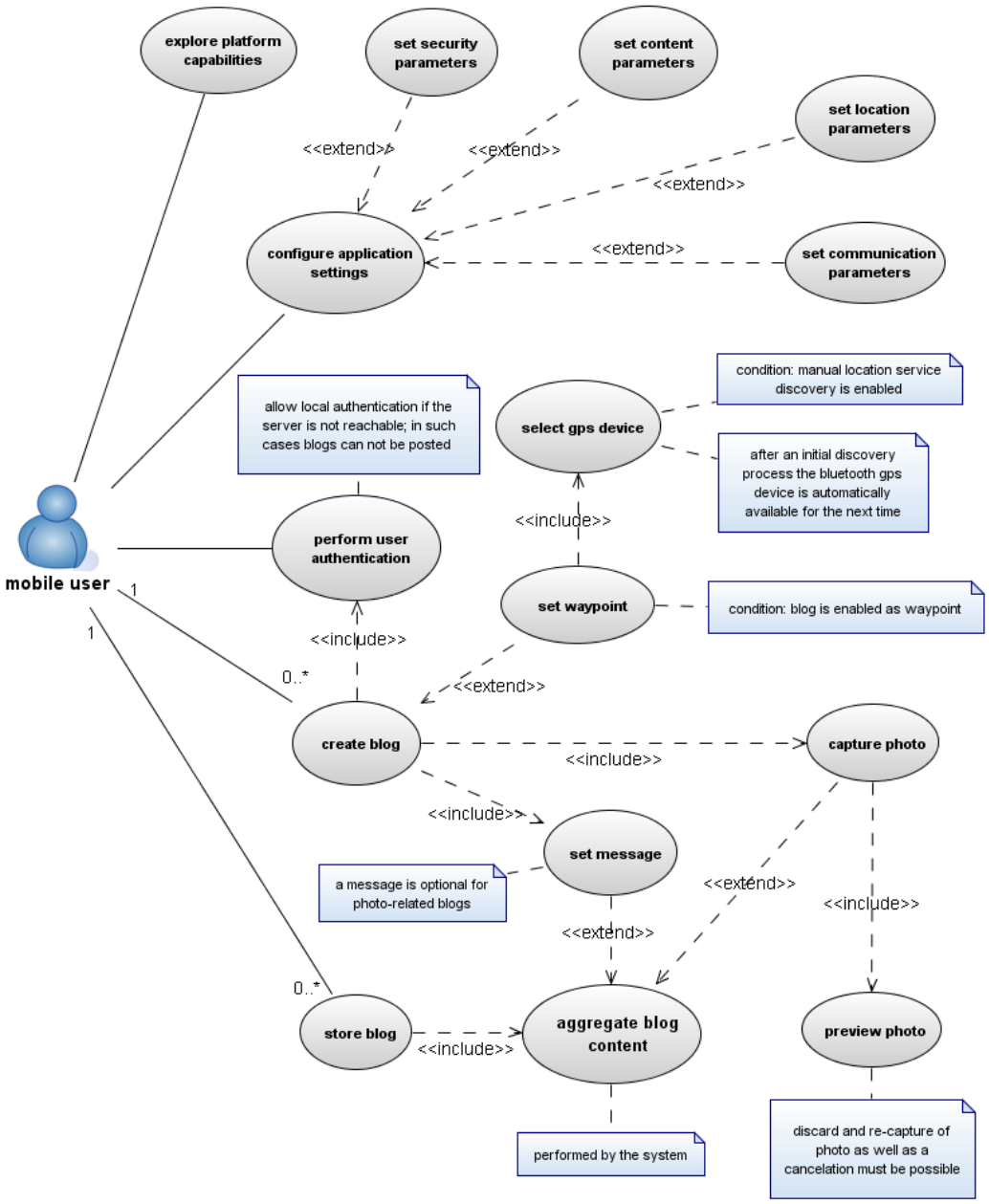


Figure 4.3.: *UML User Case Diagram for User Interactions in the Mobile Client Domain: Application Configuration and Content Generation*

4.3.1.1. System

- **Platform Capability Awareness**
  While the end user is installing and executing the MIDlet through the system's installer everything may work fine. Platform implementation failures however may prevent the MIDlet to offer all its features. Due to this fact it may be advantageous to have an application functionality which explores the actual system's firmware version and its capabilities with all available APIs to find causes for potential conflicts more efficiently. This feature is interesting for *Debugging* purposes and might be disabled easily. In

future implementations, distributed vendor-driven service platforms might give this topic however a new dimension (see section *Java Service Platform* [3.4.7]).

- **Multiple Locale**
  Since the application is based on mobility and designed for a global usage, it is obvious to support multiple languages. *J2ME Polish* enables the outsourcing of language-specific artifacts to text documents and therefore a separation from the application logic. This resource-centric web- and enterprise application development concept offers an efficient deployment of language-specific MIDlets.

- **Advanced Usability**
  Based on the features of *J2ME Polish* and application-specific ones, the software should have a high degree of usability. This involves an intuitive application navigation through advanced mobile layout features, a right-balanced usage of expressive icons, an adequate user feedback about errors and progresses of operations, and customizable units of measurement.

- **User Preference Settings**
  The user should have a great impact on the behaviour of the application. The features of the application domains *Security*, *Content*, and *Location* should therefore accept the following user-specific settings across sessions, which are discussed in more detail within the according sections.

Attribute	**Values**
`Remember Credentials`	`yes | no`
`SSL Connection`	`yes | no`
`Blog Relation`	`Journey Blogs | Standalone Blogs`
`Blog Order`	`Time | Type`
`Photo Size`	`Small | Large`
`Blog as Waypoint`	`yes | no`
`Location Service`	`Search | GPS Device`
`GPS Device`	`Search | 'GPS Device Name'`
`Distance Unit`	`Kilometers | Miles`

Table 4.1.: *Available User Preference Settings in the Mobile Client Domain*

- **Persistent Storage**
  Like with the system settings, generated content must be stored persistently to be available over successive sessions. As a mobile database the JavaME-specific *Record Management Store* (RMS) should provide this functionality since it is a native feature on all KVM devices and does not result in additionally overhead. The RMS is kept quite rudimentary and does not support database features like SQL and object-orientation. More sophisticated DBMSs for the mobile domain, like *Java DB* and *Oracle Berkeley DB* might improve the database handling.

4.3.1.2. Security

- **Authenticated Device-aware Server Access**
  Security is an important aspect when handling sensitive user-centric data. In order to

additionally prevent misuse of user accounts through unauthorized persons, credentials should be bound to a particular device. Such a strategy may be realized through the specification of a unique device identifier like the Bluetooth MAC address. This device ID should therefore be accessable within the application's login area for registration purposes. Hence the system's *Identity Management* only grants access to server features if it is aware of the user's mobile terminal. Furthermore, a user may want to let the device remember user credentials to speed-up the startup phase. This option should only be possible as soon as the user once logged-on successfully.

- **Authenticated Standalone Application Access**
  Connection instability and service unavailability should be rare, but may occur frequently especially within a mobilized environment. Environment-based *Dead Zones* within the providers telecom infrastructure (E.g. tunnels, caves, faraway sections of a country, etc.) usually prevent reliable connections. The application's design should consider this fact by ensuring an access to the application independently of a server-based logon. In such cases all non-server-based features should be made available after a successful local user authentication. A condition for the activation of this feature should also be one previously successful server logon.

- **Encrypted Transfer Channels**
  Passwords, location information, and further content should be transfered in a secure way by utilizing the SSL feature, which is part of the communication model of mobile platforms. Since encrypted connections introduce some overhead and possibly intercepted credentials are only valid for a particular device, the user should be allowed to disable this feature.

- **Logout**
  As far as the user has initiated a session with the server during the application startup, the session should be invalid as soon as the user leaves the application. It should also be ensured that the application is terminating under all circumstances, independently of an available server connection in this moment.

4.3.1.3. Content

- **Flexible Multimedia Content Generation**
  The actual business data is based on the I/O features of the device. In the prototype at least a content generation in form of *Text Messages* and *Photos* should be included. However, the design should allow a usage of the application even in the absense of a built-in camera. As a result, the composition of text messages must be possible at all circumstances, but not mandatory in combination with a photo-related blog. Text messages should have a maximum length of about 200 characters and the photos may be captured in at least two *Different Dimensions* with an aspect ratio of 1.33 by default. Each captured photo should be offered through a *Preview* functionality to be able to discard it, if it is not to the user's full satisfaction. Furthermore the software package should allow a configuration of arbitrary photo dimensions as well as either `jpg` or `png` encoding through an application configuration file. Beyond the actual prototype implementation, a possible system extension might be the integration of *Time-based Content*. This does however require additional performance measurement and related studies to access this kind of content in a reliable way from multiple platforms.

- **Journey-related Content Management**
  A stored content may be seen as *Standalone Blog* without any relationship to other blogs or it may be handled as a *Journey Blog* which is logical linked with other journey blogs. In the latter case the blog should preferable contain a unique and successive journey blog number as long at is available on the mobile platform. A *Journey* itself may be treated as a set of journey blogs and is related to a user-definable name. In this sense it is also important to design the application in a way to manage a journey as a whole, by affecting multiple blogs through single operations.

- **Content Browsing**
  The availability of the generated content on the mobile platform should be a central aspect. The blogs should be browsable in form of a list with *Expressive Blog Items*. Each blog representation should indicate the content it holds as well as the relationships among blogs. The blog items might be realized through *Flags* and icons or *Thumbnails* for photo-related blogs respectively. Furthermore, thumbnail sizes should be automatically adapted to be complied with screen properties of the underlying devices. To provide a better and more customized overview of stored blogs, the blog items might be *sorted* either by their creation date or by their representing blog type, through the user. The blog item attributes should be defined as follows [2]:

Attribute	Value
Date and Time	'Date and Time of Blog Creation'
Temporal Distance	('Temporal Distance to prev. Blog') | (-)
Spatial Distance	('Spatial Distance to prev. Blog') | (-)
Content Flags - PB	('Journey Blog Number' | -, M | -, W | -)
Content Flags - MB	('Journey Blog Number' | -, W | -)

Table 4.2.: *Blog Item Attributes for Stored Blogs in the Mobile Client Domain*

The attributes *Temporal Distance* and *Spatial Distance* in table 4.3.1.3 are discussed in section *Mobile Client Domain - Location* [4.3.1.4]). Loading a huge amount of blogs typically decreases the performance and the usability of the application. This aspect should be considered by loading the whole blog set at application startup time and by subsequently *Caching* the set, to speed-up its access when opening the list browser through a user intervention. In this case the cache must be updated each time new blogs are added at runtime. Since the application startup time increases due to this interaction improvement, it is crucial to give the user an adequate feedback of tasks which are performed by the system, already during the initialization step.

- **Selective Content Details**
  It should be possible to select and open an item of the blog list with the aim to explore more detailed information about its content. The details should be categorized according to their attribute characteristics. Related photos should be enlarged to fit them to the screen size. Available blog attribute categories should be: *Photo*, *Message*, *Properties*, and *Waypoint*. The last category needs a closer focus and is discussed in the following section *Mobile Client Domain - Location*. However, the remaining blog attributes should be:

[2] The abbreviations in the table are defined as follows: 'PB'...Photo Blog; 'MB'...Message Blog; M...Message; 'W'...Waypoint; '-'...n/a;

Attribute	Value	Category
Resolution	Large | Small('width', 'height')	Photo
Encoding	jpg | png	Photo
Content	'Text Message'	Message
Length	'Amount of Characters'	Message
Date	'Date of Blog Creation'	Properties
Time	'Time of Blog Creation'	Properties
Blog Size	'Size'(bytes | kb)	Properties

Table 4.3.: *Fundamental Blog Attributes in the Mobile Client Domain*

- **Selective Content Removal**
  Each stored blog of each type must be removable individually, whereas the allocated storage must be freed. A journey and its according blogs should also be removed at once, which however requires an additional user confirmation before the operation will be performed.

- **Content and Storage Statistics**
  The application should provide a high-level overview of stored blogs in association with their corresponding types, as well as information about the consumed and the available amount of mobile data storages and their last modifications.

4.3.1.4. Location

- **Flexible Usage of Location Infrastructures**
  The main focus related to positioning lies on the accurate and globally available *GPS-based* infrastructure (see section *Satellite-based Positioning Systems* [3.5.2]). However, additional approaches offered through *Cellular-based* location services (see section *Network-based and Hybrid Positioning Systems* [3.5.3]) should also be available, as far as they are supported by the network operator. To let the user know which positioning infrastructure has been used, the following additional blog attributes should be specified in the *Waypoint* category:

Attribute	Value	Category
Location Method	Satellite | CellID | AOA | TOA | TA	Waypoint
Hyprid Approach	yes | no	Waypoint

Table 4.4.: *Location Service-related Blog Attributes in the Mobile Client Domain*

Next to the availability of all this location methods through the standard functionality of the Location API (see section *Location-Based Infrastructures - Programming Interface* [3.5.1.4]), the prototype should also be designed in a way to retrieve the geodetic datum (longitude, latitude) directly from an external GPS receiver by accessing the BT-transmitted data (see section *NMEA Protocol* [3.1]). This customized approach does not rely on a specific API and therefore offers a higher user base. Furthermore it allows a fine-tuned access to location properties, but it also requires a manual discovery and integration of a *BT-based GPS Receiver*. As soon as a receiver has been successfully integrated once, the BT-specific connnection properties should be stored under the BT-service name within the system settings as an additional location service option. This does not require a

time-consuming initialization on each service access and allows the user to re-initiate the BT-discovery step through the system settings, if another receiver should be used. The functionality of the Location API implementation should be however enabled by default to support a maximum of positioning systems. Sometimes no location methods are available. Therefore, the blog generation process should be possible without a manifestation as a location-related waypoint. Location-unaware blogs should be transfered equally, but they must be treated in a different way at the server-side.

- **Content-related Distance and Time Awareness**
  To support *Mobile Tracking* next to the blogging capabilities, journey blogs should be aware of their spatial and temporal distance to previous journey blogs as far as they are treated as waypoints. For a better clarity, this information should be part of the browser's list items. Mobile tracking is theoretically independent of remote processes[3] and a convenient feature to trace covered distances in customized granularity. A journey's totally covered distance[4] and the journey's duration should also be offered to the mobile user. If a journey waypoint has been deleted, its logical successor (as far as existing) must update its temporal and spatial distance by using the waypoint information of the logical predecessor of the deleted one, to prevent inconsistency in the journey blog chain.

- **Content-related and logical Location Information**
  A blog's geodetic datum should be used to determine logical information of its related location on demand. This information should include the *Province Name*, *State Name*, and *Country Name* as well as the *Country's International Code*. As soon as the blog's waypoint is resolved in this manner, the according information should be accessable as a replacement for the underlying geotetic data as part of the *Waypoint* category in the blog details. This waypoint-related blog attributes might be defined as follows:

Attribute	Value	Category
`Longitude`	`'Longitude Value'`	Waypoint
`Latitude`	`'Latitude Value'`	Waypoint
`Waypoint Elevation`	`'Elevation Value'(m | ft)`	Waypoint
`Nearby Place`	`'Name of next populated Place'`	Waypoint
`State/Province`	`'Name of State/Province'`	Waypoint
`Country`	`'Name of Country'('Int.Code')`	Waypoint

Table 4.5.: *Blog Location Information in the Mobile Client Domain*

The according knowledge base for this feature is accessable through the public *Geonames* [geo08] web service. Since the elevation data within the *NMEA Sentence* is often quite inaccurate, this value should also be queried from this web service. On the one side this strategy requires an Internet connection with a resulting unavailability of the waypoint's elevation value in dead network zones, on the other side the values are very precise. The web service related elevation data origin from the NASA's *Shuttle Radar Topography Mission* (SRTM) in the year 2000. According to [geo08], *'...the dataset covers land areas between 60 degree north and 56 degree south'* and has been captured approximately every 90 meters.

[3] In the prototype implementation the elevation value however depends on a web service.

[4] Since waypoint distances are always *linear*, the resulting journey distance approximation would be quite inaccurate if the dense of waypoints is low.

4.3.1.5. Communication

- **Flexible Integration of Remote Processes**
  In principle each remotely connected or local service for the aggregation of the blog content data should be integrated into the MIDlet's application logic on demand, without a considerable execution overhead. The delivery of composed multimedia blogs and the interaction with the application server is however a more time-consuming and resource-stressing process. The performance of the underlying mobile communication threads mainly depends on the interaction models and the related protocols of the corresponding services. The application would need to interact with several different services:

  1. A *NMEA-based* location service residing on an external GPS unit.
  2. A *MLP-based* technology-independent location service.
  3. A *REST-based* web service for accessing logical location information.
  4. A *SOAP-based* web service in order to exchange data with a application server.

  The mobilized WSCs and their according remote WSPs offer a flexible loosely-coupled approch to separate the execution logic between Internet-based network nodes while enabling a convenient invocation mechanism. The usage of a web service for transfering the blogs seems to be a good decision related to text-based ones exclusively. However, a further reason for this decision is also an examination of the service model's adequacy for transfering large binary data within a mobile context (see section *Mobile Web Services - Payload Communication Constraints* [3.4.6.4]).

  If performance measurements are to a developers satisfaction, the SOA approch offers a straightforward scaling of functionality on mobile devices. *Geonames* is one example for newly upcoming services in the mobile B2C domain for this purpose (also see section *SOA Analysis and Design* [3.4.4] and *SOA Implementation and Interface Issues* [3.4.5]).

- **Selective Content Posting**
  Each stored blog of each type should be transferable individually to the server platform. It is important to find a compromise between content quality and runtime/network performance, respectively. The application should be designed in a way to reach an adequate request-response time by assuming a *Synchronous Document-centric Communication Model* and an GPRS-based transfer speed. Due to this fact, a blog should not exceed a specific size. This upper boundary must be determined during the application testing phase and configured through photo-related parameters. After a blog has been posted successfully, its item must be removed from the browser list (however not in the case of a transmission error).

- **Batch-based Content Posting**
  Journey-related blogs should be additionally allowed to be posted in one go by giving the user an adequate feedback about the transfer progress. Furthermore the user must be notified about a possibly long-term operation in advance.

- **Flexible Usage of available Wireless Networks**
  Since WLAN access points might be easily integrated through native system features on demand, this topic does not need a special focus. However, as mentioned in the beginning of this chapter, BT transfer channels might be an alternative bridging technology for the prototype. In such a case, an according BT-service would be intergrated in the same way as a device-based location service. In general, the user should also be aware of an established connection as well as about the utilized network type.

- **Awareness of Posted Content**
  After a blog has been posted successfully it should still be accessable in a further context to give the user some feedback about previously performed operations. Also detailed blog information should be available. To save storage resources, referenced full quality photos should however be removed from the RMS. Finally the user must have the option to definitively clear all blog items and corresponding RMS entries.

  This feature should enable some kind of *Pseudo Remote Browser* since the according entries are mirroring at least a subset of the blogs on the server. The blog item attributes for this remote browser should be defined as follows[5]:

Attribute	Value
Blog Type	'Message Icon' | 'Photo Thumbnail'
Date and Time	'Date and Time of Blog Transfer'
Journey Name	('Name of corresponding Journey') | (-)
Content Flags - PB	('Journey Blog Number' | -, M | -, W | -)
Content Flags - MB	('Journey Blog Number' | -, W | -)

Table 4.6.: *Blog Item Attributes for Posted Blogs in the Mobile Client Domain*

  As an improvement, a server-side service process might be used to optionally synchronize the mobile platform with the acutally posted blogs (possibly at startup time), to support a true consistency between the nodes. A further extension might be a bidirectional messaging feature to perform this synchronization directed on posted blogs, which are related to reply messages from the community. This application-centric bidirectional communication model might be based on an already proved messaging framework, involving various other communication channels like MMS, SMS, E-Mail, etc.

- **Awareness of occured Data Traffic**
  Since the usage of provider-enabled communication links is always related to charging fees, the mobile blogger should have a possibility to explore the volume of data which has been already sent over the network by the application. This value should be related to the date of the last user-initiated counter *Reset*.

[5]The abbreviations in the table are defined as follows: 'PB'...Photo Blog; 'MB'...Message Blog; M...Message; 'W'...Waypoint; '-'...n/a;

4.3.1.6. Flow Control

The following *Flow Control* diagrams illustrate the working principle of the central mobile application management components by considering the functional requirements as described above. Figure 4.4 refers to the actual blog creation process.

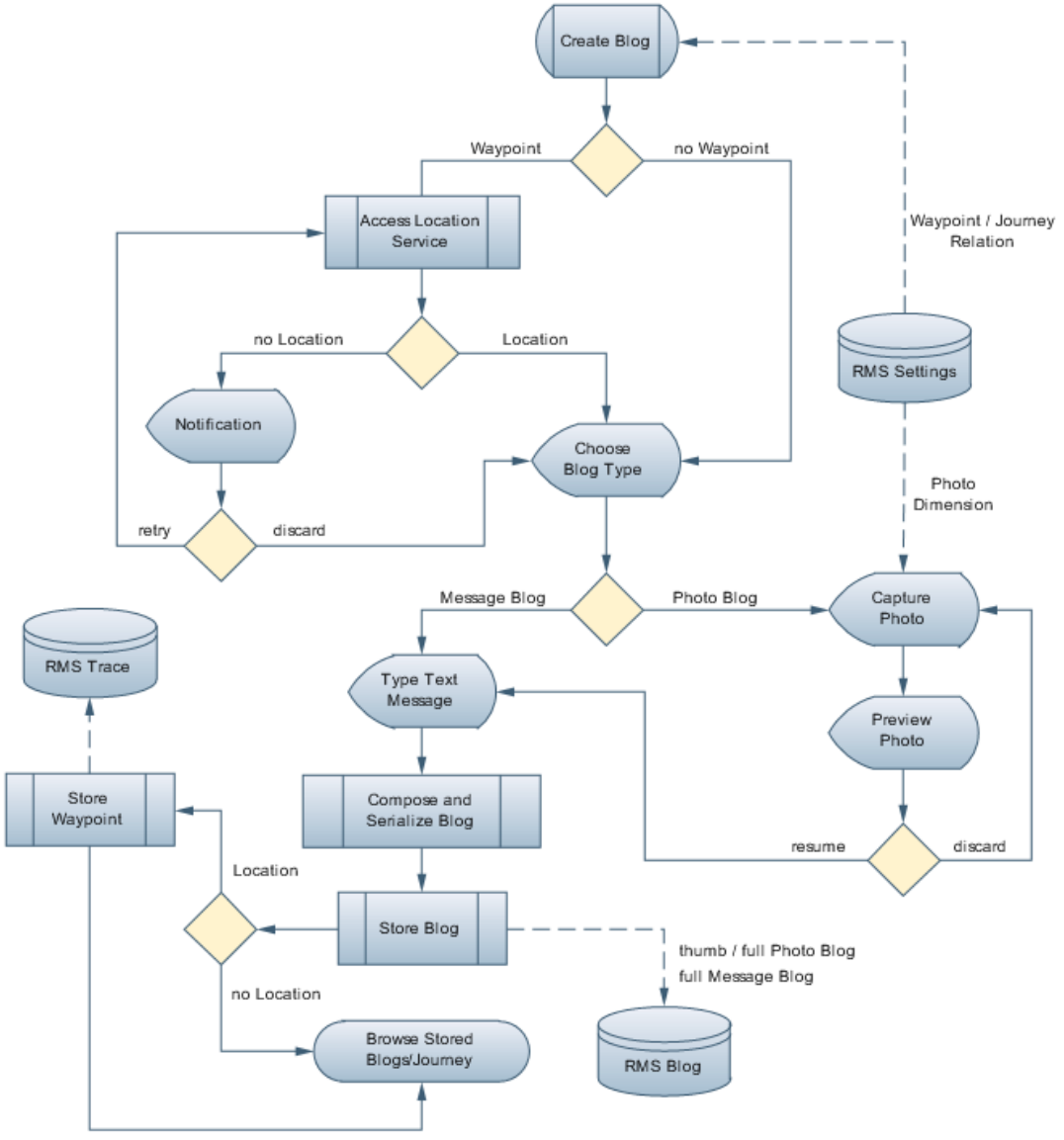


Figure 4.4.: *Flow Control Diagram for Creating a Blog in the Mobile Client Domain*

Figure 4.5 shows the flow control process for accessing blog set manipulation capabilities, whereby it is assumed that a valid user account and a related set of blogs are existing.

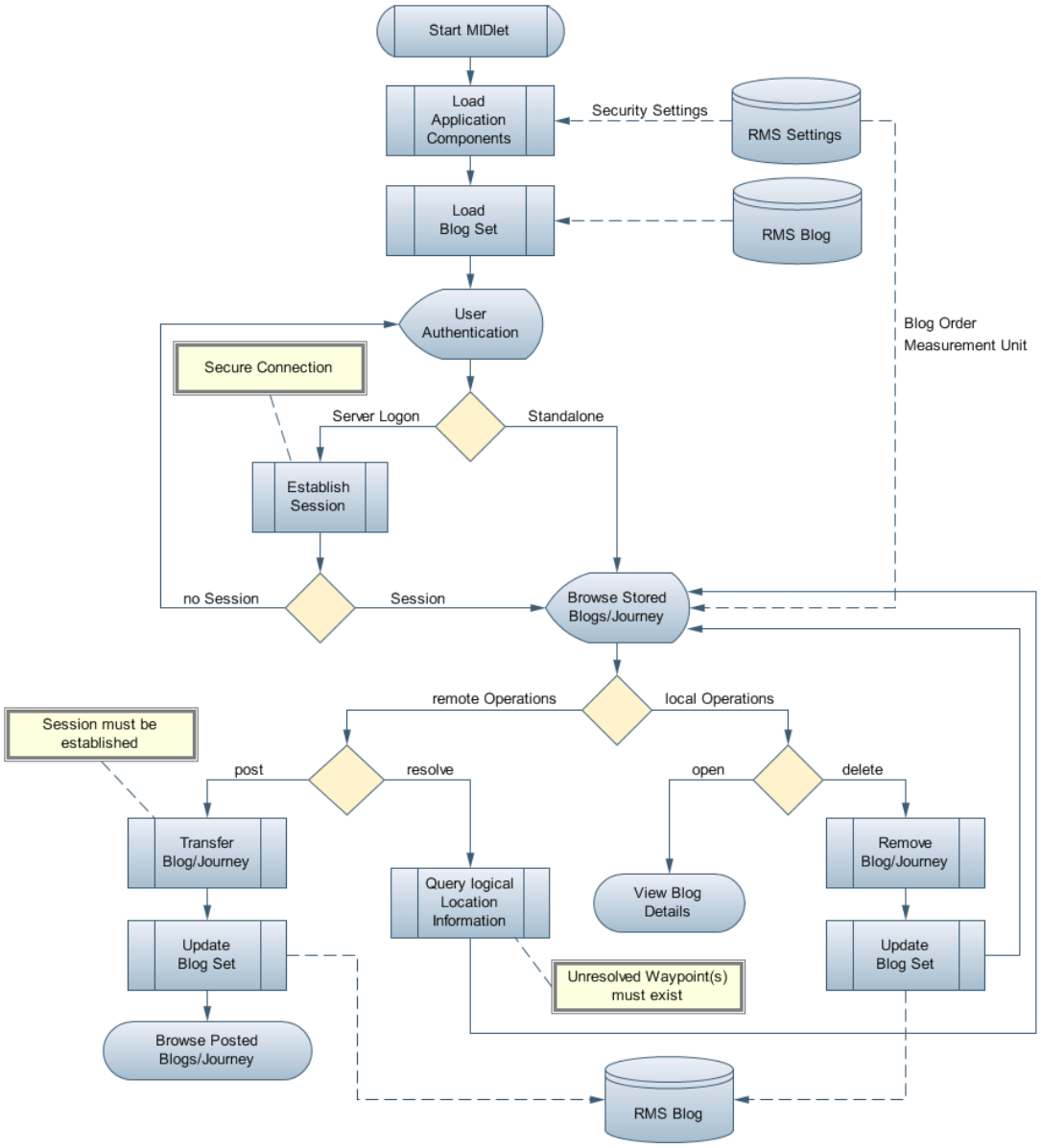


Figure 4.5.: *Flow Control Diagram for Accessing the Blog Set Manipulation Capabilities in the Mobile Client Domain*

4.3.2. Server and Backend Systems Domain

The application server with its corresponding backend system has several important functions. In principle it has to offer suitable interfaces to exchange data with the mobile terminals. Additionally, it is responsible for managing user accounts as well as to supply posted blogs to web users. Since the server only needs to have a *Servlet Container* for this purpose, no JavaEE-compliant application server is necessary. The backend system is aimed at storing and accessing user- and blog-related data. As a result, the application server contains three

major components: *Web Application*, *Blogging Service*, and *Database System*. In the following sections, specific requirements for these server features are analysed and discussed. The whole described functionality should be finally implemented as part of the prototype.

4.3.2.1. Web Application

The web application should primary offer an interface for web clients to access posted blogs in a suitable way, whereby the application should be accessable through authorized users exclusively. The according GUIs of the web components should provide an adequate feedback to the user on performed operations. To support sophisticated user input validations and a suitable web application design, the MVC-based *Struts* framework[6] should be applied. Due to this requirements, the web application has to provide an *User Account Registration*-, a *User Access Management*-, and a *Blog Content Management* component.

- **User Account Registration**
  As soon as the user has installed the MIDlet, an account registration for the device will be necessary in order to use the blogging application. Next to the standard attributes *User Name* and *Password*, the declaration of the *Device ID* is therefore an obligatory account attribute. The created account is *Universal*, since it is valid for the remote mobile logon, the local mobile login, and the web application login. Next to performing database entries for the new user, the account creation process also has to allocate some storage quota for the user's media repository on the host's HDD. For the prototype implementation, no subsequent manipulation nor a deletion of an account through the user is provided.

- **User Access Management**
  Unlike the web service component, the user access management should propagate an application-centric user authentication over a standard *Basic HTTP Authentication* through the runtime environment. Since web client-specific as well as web-form related authentication input masks in combination with this standard approach are less customizable, they would migrate logic- and presentation related web application design rules. This strategy enables a fine- tuned access and session management through the application logic. In this context, the logic has to ensure that a session becomes invalid as soon as the web user initates a logout or the according web client has been closed. The application should also take care that web clients are not re-sending user credentials[7] after a session has been closed[8]. To avoid this behaviour the application logic has to prevent clients from *Caching Pages*, which might contain web-form entries. The web user should also be aware of the current session status.

  An improvement of this component might be the introduction of *Guest Accounts*[9] for trustworthy persons to access blogs related to a particular active user. This feature would support the community idea and additionally improves the system security since credentials suitable for the mobile client domain might be separated from the ones used to access the web application.

[6] http://struts.apache.org/
[7] In form of HTTP `POSTDATA`
[8] This behaviour is usually caused through browser navigation features
[9] Possibly with restricted persmissions.

- **Blog Content Management**
  In the course of a web session, a connection to the according user's resources should be established, or refused if a logout is performed through the web user. As soon as a connection to the user's resources is available, the blogs can be offered to web clients on demand. The best strategy would be to map the necessary blog attributes to a XML representation and to stream it over the according HTTP connections. To simplify the serialization process, the *XStream* library might be used [10].

  Additionally to deliver a full set of attributes for a single blog, the *Content Management Component* should also be able to response with a list of currently available blogs including a minimum set of attributes per blog. These attributes might be seen as the blog's waypoint data. As a result, this strategy improves the application's responsiveness on initial data set requests by avoiding network traffic bursts. This improvement usually has a noticeable benefit when loading a large set of blogs with possibly many additional attributes. A further advantage of this communication model would be a separation of the blog's representation within a cartographic interface and its corresponding content. An additional feature should be used to verify, if a *Mobile Session* has been concurrently established during a *Web Session*. Such a strategy would maintain a true *Consistency* between the two client domains by delivering new blogs on subsequent requests or through a server push mechanism.

  In the prototype the only tasks of the *Blog Content Management* Component are to manage resource connections and to deliver blogs, however there might be a plenty of further possibilities. For instance, the component might be used to delete blogs or whole journeys, as well as to deliver journey related blogs or to provide statistical data for enabling elevation level schemata.

4.3.2.2. Blogging Service

The blogging service is a WSP and should have at least three interfaces to offer a *Mobile User Logon*, a *Mobile User Logout*, and a *Mobile Gateway*. In SOA terminology, the functionality of this service should be an instance of a *Basic Service* consisting of logic- and data-centric elements, which does however not need to have a manifestation in further services, due to a manageable complexity of the application. The mobile application must be notified about the service operation results under all circumstances, to give the WSC and thus the mobile user an adequate feedback.

- **Mobile User Logon**
  As mentioned in section *Mobile Client Domain - Security* [4.3.1.2], remote mobile terminals need to offer a unique device ID. The according service interface only needs to accept this single parameter, because a successful authentication through the security features of the runtime environment should be the precondition to delegate control to the service.

- **Mobile User Logout**
  Since the WSP should not track IP addresses of mobile terminals, the mobile application has to pass the device ID once again as interface parameter to terminate a server session.

[10] http://xstream.codehaus.org/

- **Mobile Gateway**
  As soon as the user is authenticated, the mobile terminal should be allowed to transfer blogs over a corresponding gateway to be persistently stored on the server as a subsequent step. The according interface needs to accept serveral blog-specific attributes as parameters.

Next to providing *Communication Interfaces*, the WSP's logic should offer the following logic- and data-centric components: *Identity Mangement*, *Blog Content Handler*, and *Persistence Management*.

- **Identity Management**
  One task of this component should be a verification of the relationship between the received device ID and the user credentials before granting access to further service features. If this security instance has been passed, a *Session* has been established and the according mobile user should be declared as *Online*. As soon as the WSP receives a logout notification from a mobile user, the component should also invalidate the corresponding session by resetting the mobile user's online status. If there is no activity at the gateway interface for one hour a timeout mechanism should invalidate a user session automatically.

  Unlike as with enterprise-based identity infrastructures (see section *Mobile Web Services - Frameworks* [3.4.6.2]), this approach is straightforward, but efficient in this context since no further services are involved.

- **Blog Content Handler**
  As mentioned in section *Mobile Web Services - Payload Communication Constraints* [3.4.6.4], it seems to be good strategy to process binary data like photos as *Base64* encoded strings. In this way photos can be passed as parameters like the descriptive one. The component therefore also has the task to composite the received blog content to be suitable for further server-side processing. The reconstructed photo should be available as user-specific resource within a *Repository* on the server.

- **Persistence Management**
  Each remote invocation of service methods is affecting persistence infrastructures. The *Identity Management* as well as the *Blog Content Handler* component will need to perform I/O operations on a database and the host's file system respectively, to verify users as well as to store user-related blogs. A separate data-centric component should therefore provide a transparent interaction with this external system resources.

4.3.2.3. Database System

The application's data should rely on a RDBMS to allow a flexible access and manipulation through data-centric server processes and through the authentication module of the underlying execution environment. In principle, the corresponding tables belong to one of the following logic categories:

- **Security**
  This category should include the entities `user`, `userrole`, and `trackeduser`. The first ones should be used to associate registered users with their device and the permissions

they have on the system. The last one might be used to enable a session management by tracking all mobile users which are currently online. The usage of a separate table for this purpose would tolerate temporary server drop-outs without affecting established sessions with mobile terminals. Since a logout notification during drop-outs would not update the according entries, some session-timeout or -update mechanism would be required as well.

- **Content**
  This category should include the entities `blog` and `photo`. The first entity should store descriptive blog attributes and the second one blog-referenced photo properties. These properties should at least contain the photo's dimension[11] and a reference to their actual location in the user's repository.

4.3.3. Web Client Domain

The web application interface should offer a user interaction with the web application as well as with a further web service to integrate map-related functionalities. Through this interface the web user has the possibilities to create a new account for the blogging application as well as to access and to explore the content of posted blogs in several ways. The following figure visualizes all desired user interactions in form of a *UML User Case Diagram*.

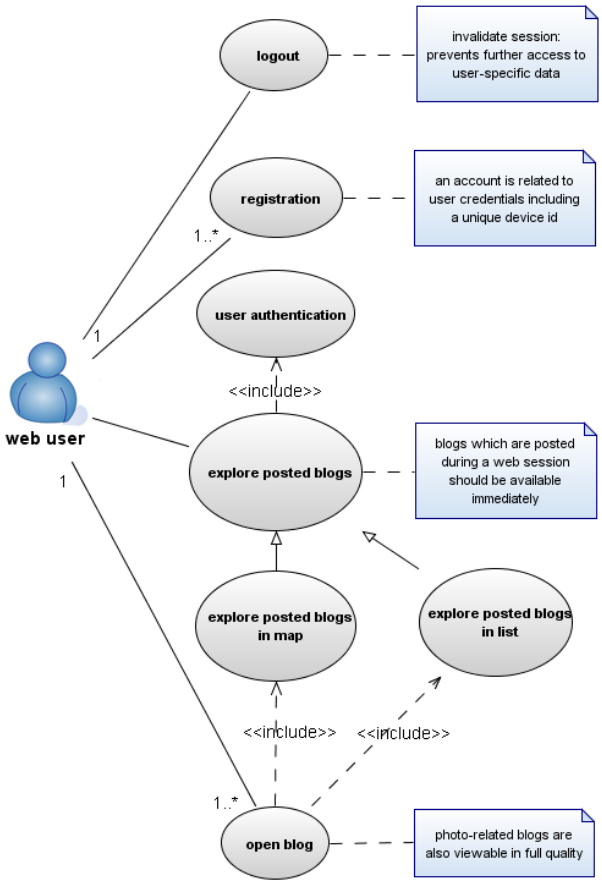


Figure 4.6.: *UML User Case Diagram for User Interactions in the Web Client Domain*

[11] Which may be of interest for their web-based representations.

The logic of the web application interface includes the components *Blog Integration* and *Blog Exploration*, which should be realized through the functionality of *JavaScript* directives.

- **Blog Integration**
  After a successful web login, all available waypoints and related blogs should be loaded from the server to be represented as GUI-based items. To support a near *Realtime Update* of newly posted blogs, the server's consistency feature should be used, by schedule requests on fixed intervals. To have a suitable communication model and to provide a stable GUI, blog request should be performed asynchronously through the browser's *AJAX* functionality. The component has to unserialize the XML-based responses into objects, in order to handle waypoints and blogs through further components. The prototype does not load blogs without waypoint information. This subject might be a part of further interaction extensions.

- **Blog Exploration**
  Loaded blogs should be represented as *List* entries through a unique number and their creation date. Additionally, the blogs should have a location-based representation within a *Map* as visual *Pins* related to the blog's *Waypoint*. The related list and map entries should be associated with each other to enable a consistent and intuitive user interaction. On the same web page textual information about the currently selected or focused blog should be available in form of the following attributes.

Attribute	**Values**
`Journey Name`	`'Name of Journey'`
`Photo`	`yes | no`
`Message`	`yes | no`
`Elevation`	`'Elevation Value'(m) | n/a`
`Longitude`	`'Longitude Value'`
`Latitude`	`'Latitude Value'`

Table 4.7.: *Textual Representation of Blog Attributes in the Web Client Domain*

The cartographic interface should be provided by the *Google Maps* web service. Its functionality is accessable through *JavaScript* functions, embedded into the web client's logic. As soon as a blog has been opened through the web user by selecting the list- or map-related blog representation, its content should be offered in a suitable way, including a photo preview as far as available. Google Maps and similar services usually provide many features to enhance their standard functionality. For instance, journey blogs might be grouped and visualized in different meaninful colors within the map.

4.4. Communication Models

Based on the application's infrastructure, as depict in figure 4.1, and the analysed functional requirements, the communication models between the application domains are fully defined. Figure 4.7 shows this inter-domain data exchange patterns by considering specified payload and security issues. A fourth domain is involved in order to access geographical information from an independent ISP. The ISP's data sources are used to resolve waypoints as described in section 4.3.1.4.

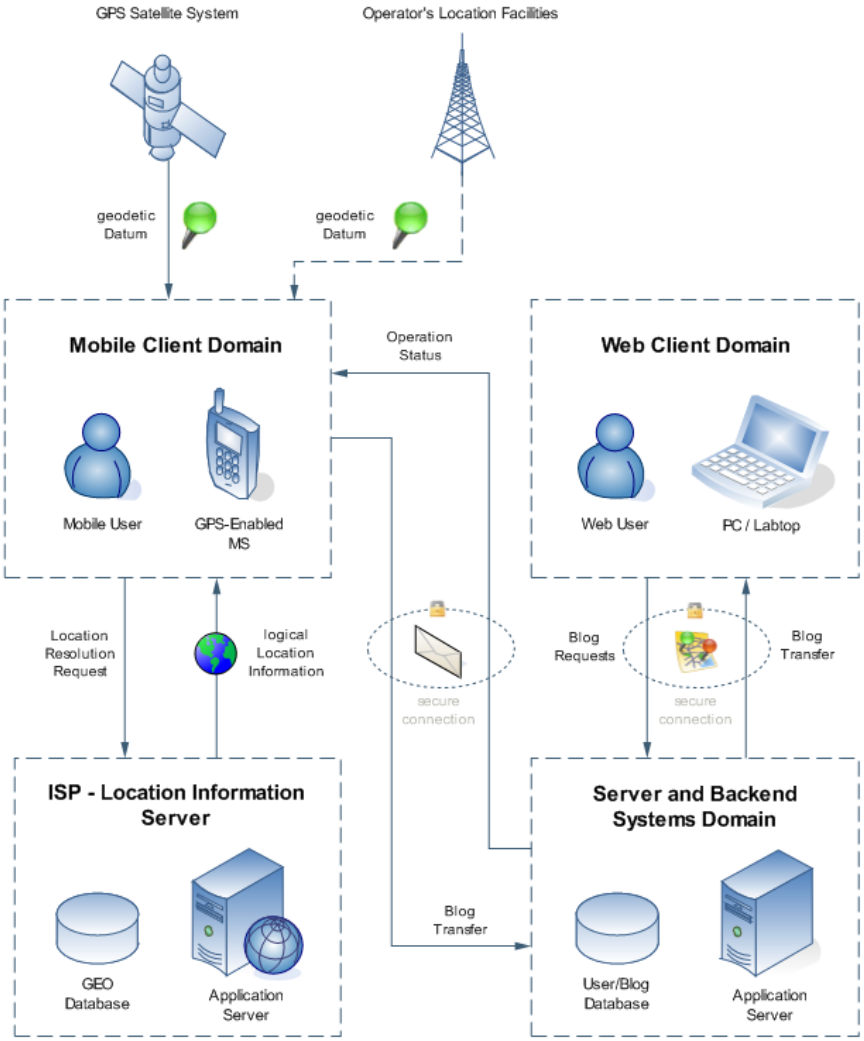


Figure 4.7.: *High-level Overview of the Blogging Application's Communication Models. The Data Exchange between the Clients and the Application Servers is based on XML-structured Messages. This Strategy supports a technology-independent Payload Handling.*

5. Prototype Design

"Nothing is paticularily hard if you divide it into small jobs."

— Henry Ford (1863-1947)

The specified *Functionality* and *Communication Models* in chapter *Requirements* [4] are the prerequisites for the design of the actual software architecture and the relationship to the utilized services. In this phase the inter-domain *Software Subsystems* and the according *Software Components*, as well as *Time-critical Communication Issues* are going to be specified in detail. As a central functionality, the mobile core logic also refers to its corresponding *Java APIs*.

5.1. Application Architecture

The following figure 5.1 is a scratch of the system's software architecture including the underlying platforms and related communication channels.

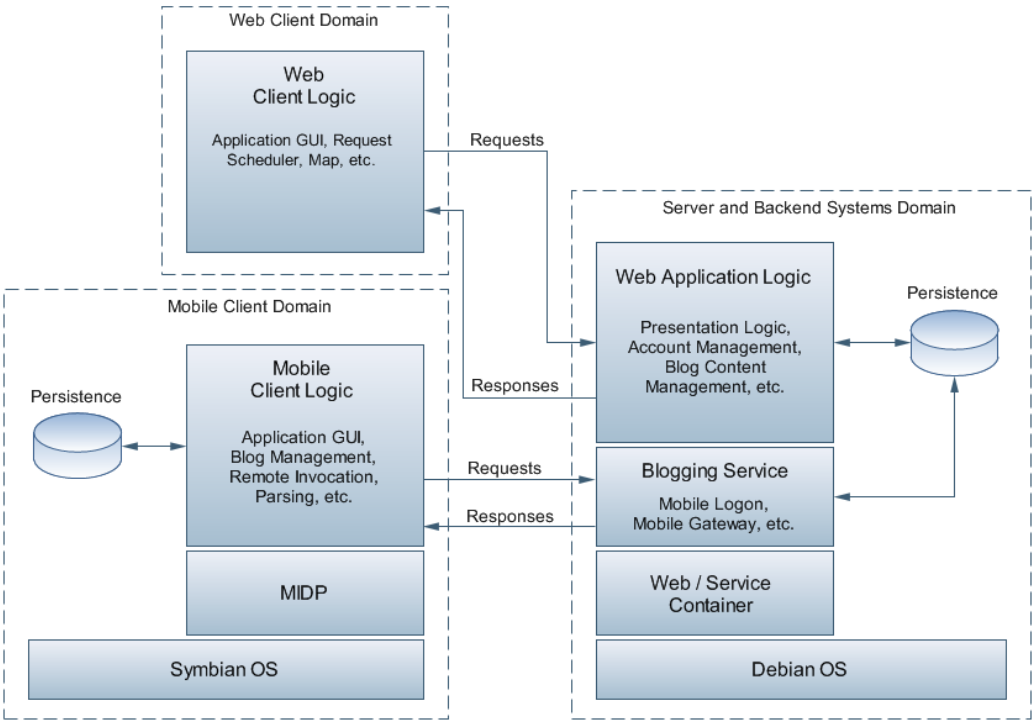


Figure 5.1.: *High-level Overview of the Blogging Application's Architecture.*

The application is designed to work in each mobile OS environment which runs the KVM. Since the prototype's *Mobile Software Components* are executed under *Symbian OS* (see section *Symbian OS Platform* [3.1.2]), the architecture refers to this platform. The server host is running the Linux distribution *Debian Etch* to offer web- and service runtime environments for *Static Software Components*, accessable through the clients. Debian is a proven light-weight OS for server machines with nearly no limitation in customizing platform features.

5.1.1. Mobile Software Components

The *MIDlet* is the central element of the blogging application. As seen from the analysis phase, the mobile client domain specifies the most functional requirements of the overall system. As a logical result the mobile implementation requires a sophisticated architecture to be able to handle the application development and possibly subsequent extensions in a suitable way. Figure 5.2 refers to the MIDlet's high-level architecture and its related subsystems.

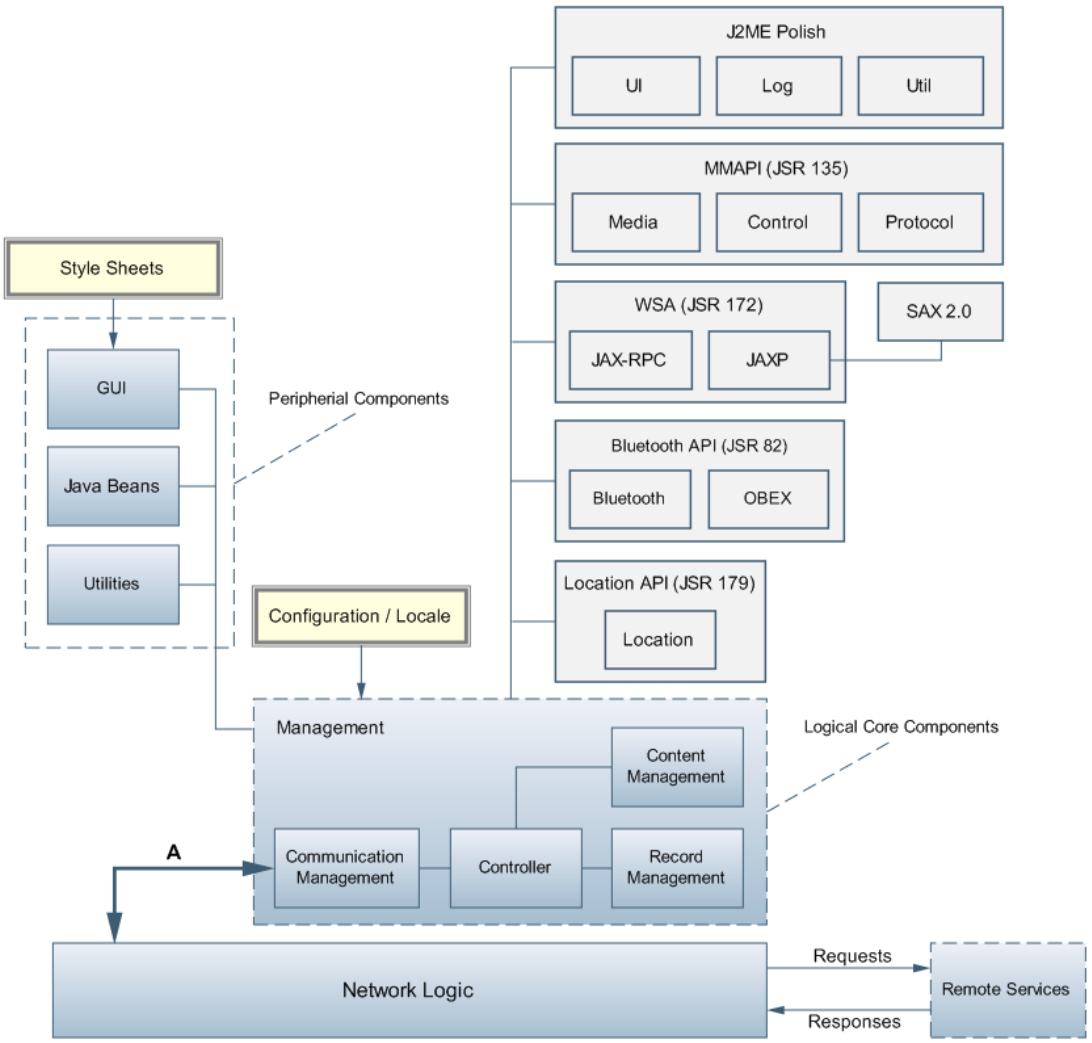


Figure 5.2.: *Mobile Application Architecture with a high-level Focus on involved Subsystems*

As depict in figure 5.2, the *Management* domain is separated into the following subsystems: *Controller*, *Content Management*, *Record Management*, and *Communication Management*. The *Communication Management* system provides access to the network logic and therefore to remote processes and data. The communication link (**A**) enables BT and HTTP connections.

Due to performance reasons, the deployment of design pattern in the mobile domain should be rare (see section *Mobile Software Development* [3.2]). Depending on the nature of the application, a deployment of the *Model-View-Control* (MVC) software development pattern might however be a good idea for comprehensive mobile implementations. Since the charateristics of a blogging application requires plenty of user- and data-centric elements, it is obvious to structure the MIDlet according to this paradigm. Furthermore, the *Singletone* pattern is used to prevent multiple instances of controlling components. Based on this design decisions, the MIDlet might be seen as a set of logical-related system components:

- **Logical Core Components**
  Functional elements within the subsystems of the management domain.

- **Utility Components**
  Assisting functional elements for the execution of logic components. E.g. parser, formatter, encoder, connection handler, etc.

- **Data Components**
  Container elements for holding and transporting application data in form of *Java Beans*.

- **Screen Components**
  Functional GUI elements for supporting a user interaction with the application. E.g. form, list, video stream, etc.

- **Resource Components**
  Configuration as well as *Style Sheet* files to define the application's behaviour, including their *Locale*-specific message elements and the screen layout.

The application involves *J2ME Polish* and several optional JSR-specific APIs to access native platform features as indicated in figure 5.2. The *J2ME Polish Building Framework* enables a flexible building process of the MIDlet through pre-defined and customized *Ant Tasks*, as well as the integration of application resource components.

In order to understand the tasks of the *Locical Components*, it is necessary to analyse the according subsystems. Figure 5.3 gives a deeper insight into the MIDlet's core architecture.

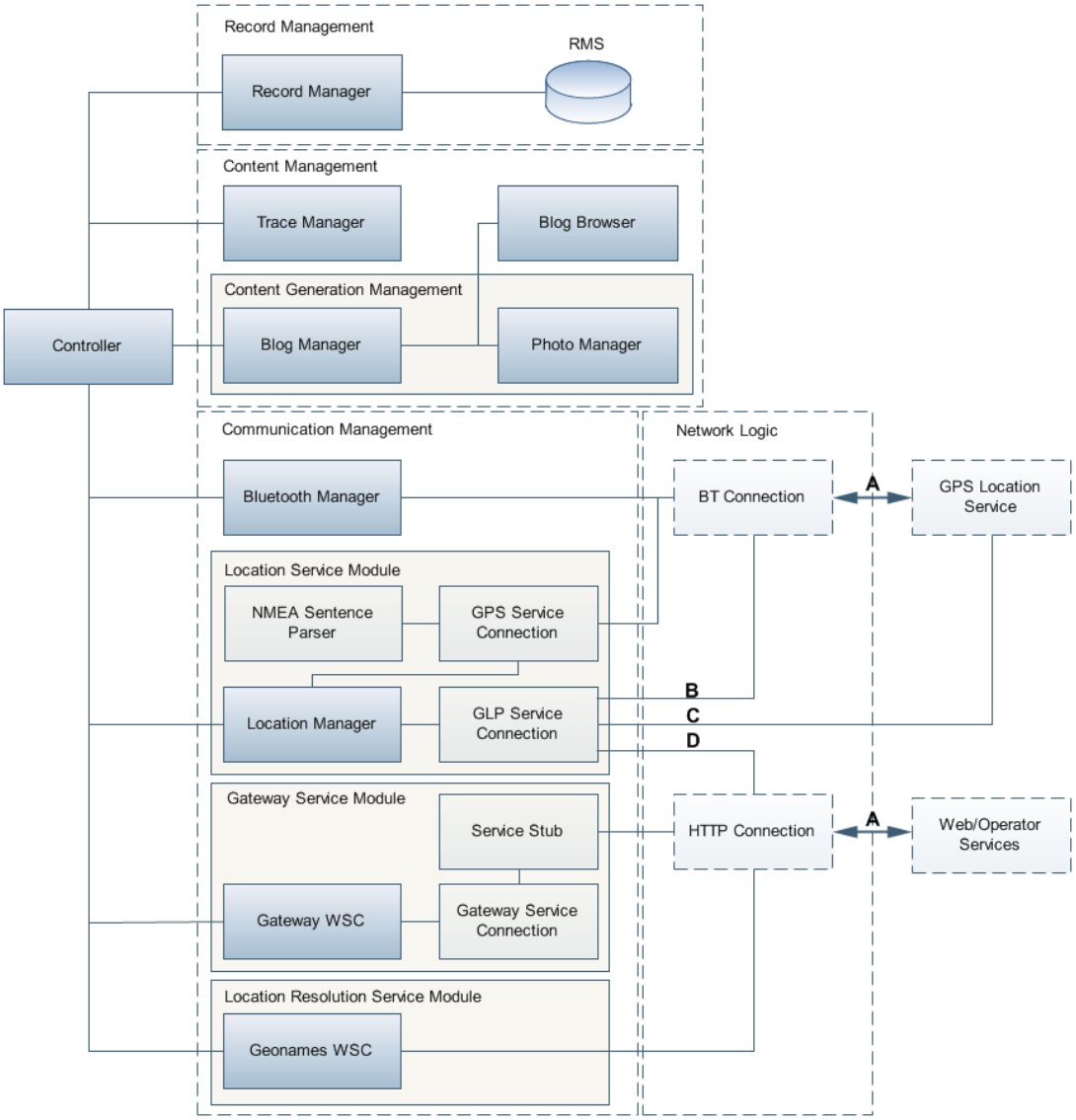


Figure 5.3.: *Mobile Application Architecture with a Focus on the Core Logic*

The *Controller*, the *Record Management*, and the *Content Management* subsystem do not rely on external resources. Each remote connection is established through the components of the *Communication Management* system by utilizing the related *Air Interfaces* (**A**). The `GLPServiceConnection` utility includes the ability to discover location services on various hardware platforms. The component might establish a service connection to an external GPS receiver (**B**), to an embedded GPS receiver (**C**), or to the network operator (**D**).

Within the *Communication Management* system the components are further clustered as *Location Service Module*, *Location Resolution Service Module*, and *Gateway Service Module*. The granularity of the *Logical Components* within the MIDlet's core architecture is suitable enough for realizing their embedded functionalities as concrete *Java Classes*. These classes represent the *Logical Core* of the mobile application. In the following only these *Logical Core Components* are discussed in more detail since they are the most important ones in the mobile client domain.

5.1.1.1. Logical Core Components

This section describes the application's *Logical Core Components* as they are implemented in the MIDlet suite, and their responsibilities and relationships among each other. Furthermore, each class refers to its underlying API specification, which considers the functional requirements as defined in chapter *Prototype Analysis* [4].

- **Controller**
  The `Controller` extends the MIDlet class, and as the starting point it coordinates the application's flow control. The component is responsible for the startup process and it therefore acts as a loader for further control components, system settings, application screens, alerts, icons, etc. See appendix *MIDlet Core APIs - Controller* [D.1].

- **Blog Manager**
  The `BlogManager` is a central element, involving essential functionalities to perform operations on user-generated content. In principle this component manages the *Blog Creation* process by composing all related content. As a first step, it gathers the device's location if a blog should be referenced to a waypoint. Depending on the type of the blog, it furthermore starts the photo capture process as well as the message input form. The involved `PhotoManager` runs within a threaded environment and notifies the `BlogManager` as soon as a photo has been created by the user. In a final step, the composed blog is serialized to be compatible with the persistence infrastructure on the device.

  The `BlogManager` is also responsible for initializing the browsing capabilities of blogs by involving the abstract `BlogBrowser` component. According to the instructions of the browser's logic, the manager manipulates a blog set in the desired way. The represented blogs data in the browser's list either refer to the locally stored data set or to the already posted ones. Finally the *Blog Manager* acts as a data provider for blog statistics. See appendix *MIDlet Core APIs - Blog Manager* [D.2].

- **Photo Manager**
  The `PhotoManager` is responsible for the photo capture, formatting, and preview processes. The *MMAPI* supports this tasks by enabling for instance an access to the device's camera. The photo encoding, the quality, as well as the dimension parameters as they are handled by the system, are relying on the application's configuration file. This strategy enables a straightforward adaption of the visual blog content and the related memory consumption. See appendix *MIDlet Core APIs - Photo Manager* [D.3].

- **Trace Manager**
  The application's `TraceManager` is used to manage relationships between journey blogs as they have been described in the last chapter. A journey might be related to a *Trace*, which is in fact an ordered list of waypoints. Such a trace is stored and processed independently of blog sets. The `TraceManager` is responsible for temporal and spatial distance measurements and it is furthermore used as a data provider for journey statistics. The `BlogManager` takes care about the consistency between journey blogs and waypoints as far as a blog set has been manipulated. See appendix *MIDlet Core APIs - Trace Manager* [D.4].

- **Record Manager**
  The `RecordManager` represents the intermediary element between the record store and

classes which need to read or write data as part of their embedded logic, including system settings as well as user-specific content persistence. Data is stored as byte arrays exclusively. By default, instances of the JavaME-specific interfaces `RecordFilter` and `RecordComparator` are used to enumerate and to process stored records. In this context it must be considered to avoid a high frequently RMS access, since the records must be temporarily reconstructed as objects followed by subsequent re-serializations, in order to identify and process according attributes. A sophisticated application design might decrease the memory footprint dramatically, even if persistence is a central element. The prototype contains the following record stores: `loginRS`, `settingsRS`, `traceRS`, `blogPMRS`, `blogPMThumbRS`, and `blogMRS`[1]. See appendix *MIDlet Core APIs - Record Manager* [D.5].

- **Blog Browser**
  The `BlogBrowser` is a generalization of the `BlogBrowserLocal` and the `BlogBrowserRemote` components. These two browsers offer the interface for the management of stored and posted blogs directly on the device. The user interface logic is suitable for an exploration of blog details, a resolution of waypoints, a deletion of blogs/journeys, an initialization of blog/journey transfer processes, and a customization of the blog list behaviour. A blog caching and updating mechanism is designated for both instances. See appendix *MIDlet Core APIs - Blog Browser* [D.6].

- **Bluetooth Manager**
  The `BluetoothManager` enables the management of surrounding bluetooth-based services. It is responsible for the discovery and the inquiry of devices and services as well as for their accessability. The component is based on the functionalities of the *Bluetooth API* and used to connect GPS receivers for location determination purposes. See appendix *MIDlet Core APIs - Bluetooth Manager* [D.7].

- **Location Manager**
  The `LocationManager` composes the capabilities of system features to access location services. For a customized GPS-based location determination, the GPS receiver is accessed through the `GPSServiceConnection` handler. The received NMEA sentence must be parsed with the `NMEASentenceParser` utility in order to get some useful geodetic data. A *Generic Location Provider* (GLP) is supported through the related `GLPServiceConnection` handler by incorporating the *Location API*. The location data delivered from an GLP is directly accessable. See appendix *MIDlet Core APIs - Location Manager* [D.8].

- **Gateway WSC**
  The `GatewayWSC` component enables high-level functions to invoke remote server-based processes for the integration of mobilized content into a web environment. It offers user authentication and blog transfer features on the device. The WSC runs in a threaded environment and involves SOAP-based data transfer functionalities through the utilization of the according `GatewayServiceConnection` handler, which furthermore relies on the actual `ServiceStub` class. The stub and the related interfaces are generated by the IDE, based on the descriptions of the service's WSDL file. Since the component utilizes the *WSA*, the actual invocation and the XML-parsing tasks are managed by the system. See appendix *MIDlet Core APIs - Gateway WSC* [D.9].

[1] PM refers to multimedia (photo, message) blogs; M refers to message blogs

- **Geonames WSC**
  The `GeonamesWSC` component is responsible for enabling logical location information for a waypoint's geodetic data. The public *Geonames* service provides information like the *Country Name*, the *NearbyPlace Name*, and the *Elevation Value*. The component runs in a threaded environment and is based on a REST-based service invocation model. The response must be parsed manually by accessing the *SAX Parser* features on the device. See appendix *MIDlet Core APIs - Geonames WSC* [D.10].

5.1.1.2. Peripherial Components

The core logic described in the last section, is strongly related to further components in order to get a runable prototype with the desired functionalities. These peripheral components have important functions on all application levels and are related to specific domains within the mobile application.

Component	Domain
`Settings`	System
`PlatformInfo`	System
`Credentials`	Security
`Blog`	Content
`BlogMessage`	Content
`BlogMultimedia`	Content
`Photo`	Content
`Message`	Content
`GPSLocation`	Location
`Waypoint`	Location
`Toponym`	Location
`Payload`	Communication

Table 5.1.: *Data Components in the Mobile Client Domain*

Component	Domain
`AppSplashScreen`	System
`AppLoginScreen`	System
`AppMainScreen`	System
`AppSettingsScreen`	System
`PlatformInfoScreen`	System
`PlatformMainScreen`	System
`BlogBrowserScreen`	Content
`BlogMainScreen`	Content
`BlogMessageScreen`	Content
`BlogInfoScreen`	Content
`PhotoCaptureScreen`	Content
`PhotoViewerScreen`	Content
`BluetoothDeviceList`	Communication

Table 5.2.: *Screen Components in the Mobile Client Domain*

The *Data-* and *Screen Components* in table 5.1 and 5.2 are used for data processing, inter-component communication, and user interaction purposes. The screen components as well as different types of system alerts are managed and accessed through the application *Controller*.

The *Utility Components* as listed in table 5.3 are essential to manage complexity by encapsulate frequently used functionality within the different domains and to support code maintainability through interface components.

Component	**Domain**
`Monitor`	System
`Statistics`	System
`DateFormatter`	System
`ActivityAlert`	System
`InfoCollector`	System
`BluetoothInfoCollector`	System
`DisplayInfoCollector`	System
`FileConnectionInfoCollector`	System
`LibrariesInfoCollector`	System
`PlatformInfoCollector`	System
`ISerializable`	Content
`BlogSerializer`	Content
`BlogUnserializer`	Content
`BlogInfoFormatter`	Content
`Base64Encoder`	Content
`RMSBlogSelector`	Content
`GPSServiceConnection`	Location
`GLPServiceConnection`	Location
`NMEASentenceParser`	Location
`NMEASentenceParserToolkit`	Location
`WaypointToolkit`	Location
`RMSWaypointSelector`	Location
`GatewayServiceConnection`	Communication
`ResponseHandlerCountry`	Communication
`ResponseHandlerNearbyPlace`	Communication
`ResponseHandlerElevation`	Communication
`BluetoothDevice`	Communication
`IServiceListener`	Communication
`IGateway`	Communication

Table 5.3.: *Utility Components in the Mobile Client Domain*

5.1.2. Static Software Components

The *Application Server* involves several subsystems to handle the complexity of managing and processing client requests from different domains. Figure 5.4 refers to the application server's architecture and its related software components.

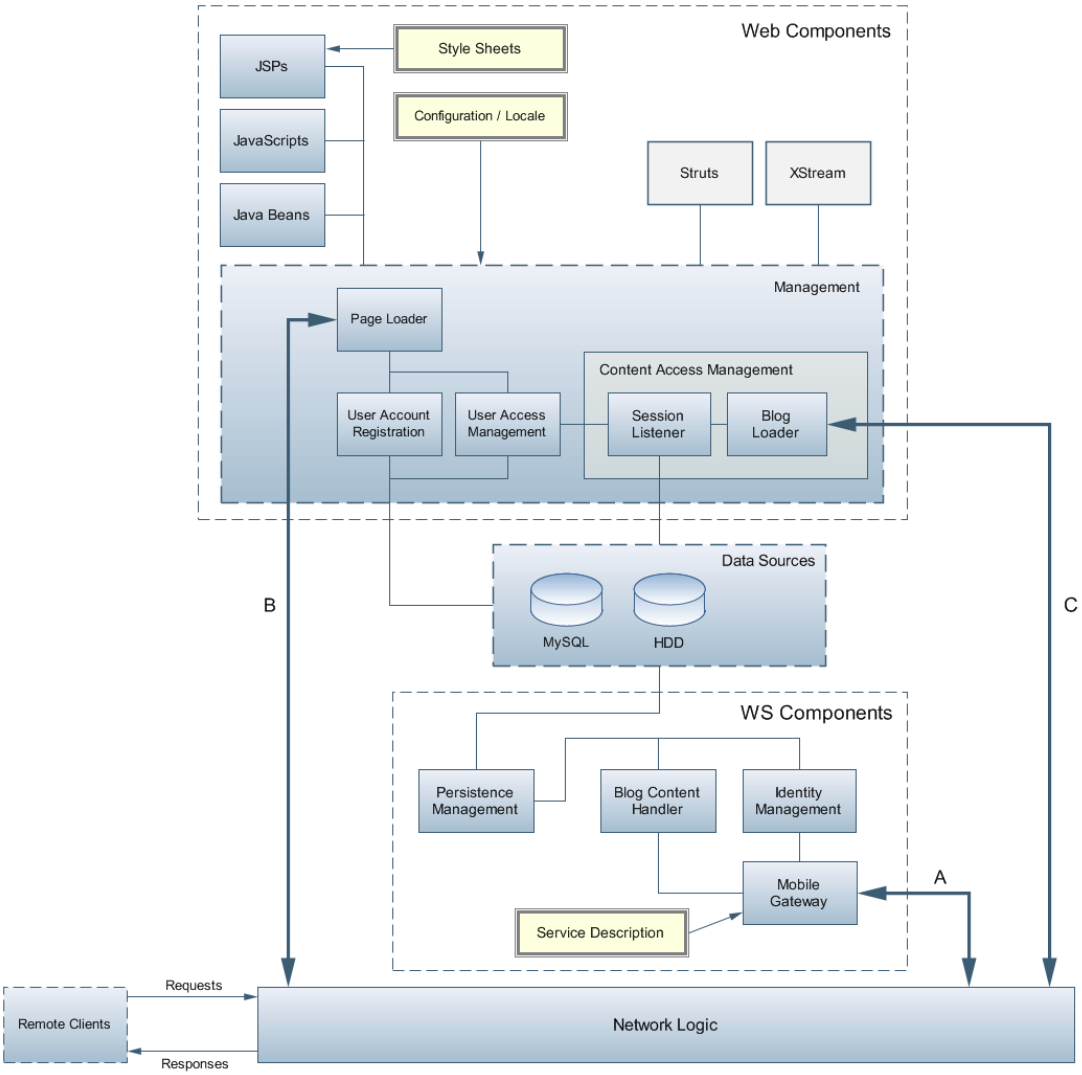


Figure 5.4.: *Web Application- and Blogging Service Architecture*

The two server domains *Web Application* and *Blogging Service* are going to be observed under the viewpoint of software components. The server architecture specifies *Web Components* and *Web Service Components*, which are operating in different server environments. They are only coupled by means of manipulating and accessing shared application *Data Sources*.

In the same way as with the MIDlet, the *Web Application* components are structured according to the MVC software pattern. This is obvious since the *Struts* framework implies an architecture of the *Web Components* as follows:

- **Logical Components**
  Functional elements within the subsystems of the management domain.

- **Data Components**
  Container elements for holding and transporting application data in form of *Java Beans*.

- **Screen Components**
  Functional GUI elements based on HTML pages for supporting a user interaction with the web application. E.g. form, list, button, etc.

- **Resource Components**
  Functional elements for encapsulating the web client logic in form of *Java Script* files and *Configuration-* as well as *Style Sheet* files to define the application's behaviour and the page layout.

Since the intermediary *Web Service Components* are functional components only, no end user interface is necessary. The *Blogging Service* is structured into:

- **Logic-Centric Service Components**
  Functional elements for handling identities and the received data from connected service consumers.

- **Data-Centric Service Components**
  Intermediary functional elements for managing the access to persistent infrastructures on the host.

- **Data Components**
  Container elements for holding and transporting service data in form of *Java Beans*.

- **Resource Components**
  The service contract in form of a *WSDL* file. According to the interface definitions, the stubs are typically generated through IDE tools by requesting the contract of a known service endpoint.

As shown in figure 5.4, both server domains have to establish communication links to their clients. The *Blogging Service* communicates with the mobile clients by exchanging *SOAP* messages in a synchronous manner (**A**). The *Web Application* communicates with the web clients in order to deliver HTML pages through a synchronous channel (**B**) or application-specific blog content through an asynchronous channel (**C**). In the following sections the most essential components of both domains are briefly described. As with the MIDlets's logical core components, the functionality of the server-specific ones are equally suitable for a manifestation into *Java Classes*.

5.1.2.1. Web Components

- **Page Loader**
  The abstract `PageLoader` component is a *URI Handler*, responsible for the delivery of web resources as well as a request dispatcher for the delegation of form-specific user

data to the `UserAccountRegistration` and `UserAccessManagement` components. This process in mainly driven by the *Struts* framework. The usage of *Tag Libraries* within *Java Server Pages* (JSPs) enables functional *Widgets* for HTML-based interaction patterns like a straightforward rendering of error messages through *Struts Action Errors*. The JSPs communicate with the application logic typically through *Java Beans*. This data-transfer process is automated by including *Struts Action Forms*[2] as well as the functionality of customizable user input *Validators*. Finally, the *Struts Action* components include the actual application logic. Additionally, *Struts Action Forwards* might be used to decouple JSP's from the *Action* components.

- **User Account Registration**
  The logic for the user registration process is embedded into a corresponding *Action* component. As soon as the user's *Registration Data* has been validated successfully, the component creates a new account by setting the according entries into the database tables `user` and `userrole`. Furthermore, HDD storage quota is allocated for the account by creating a subfolder within the server's `repository` directory. Finally, the component induces a loading of the application's main page.

- **User Access Management**
  The logic for the user access management process is embedded into a corresponding *Action* component. As soon as the user's *Login Data* has been validated successfully, the component authenticates the user by verifying the given credentials with the entries in the database table `user`. If access is granted, the component enables a *Web Session* with the client by using the session management functionality of the *Servlet API*. Finally, the component induces a loading of the application's main page. On a *Logout* request the component ensures an application-wide unavailability of sensitive user data on subsequent requests, by closing the according active session. To prevent clients and proxies from caching pages, according HTTP directives in the HTML headers must be set.

- **Session Listener**
  As a subcomponent of the *Content Access Management* subsystem, the *Session Listener* uses the `HttpSessionBindingListener` functionality of the *Servlet API*, and it is used to dynamically establish and close database connections according to a user's session status. This component is a interface to the user's blog set, and a loader as well as a composer of blog- and waypoint lists by accessing the actual blog content through the database table `blog`. Furthermore it is responsible to observe the user's mobile status to enable blog consistency during active sessions between clients, by verifying entries in the database table `trackeduser`.

- **Blog Loader**
  As a further subcomponent of the *Content Access Management* subsystem, the *Blog Loader* is a simple *Servlet* for handling blog requests. It incorporates the related `SessionListener` in order to fetch blogs and to verify the user's session status. The `BlogLoader` component uses the *XStream* tool to map the according *Java Beans* into *XML-Representations*, suitable for the network transfer.

[2] This components might be seen as special *JavaBeans*.

5.1.2.2. Web Service Components

- **Mobile Gateway**
  Since a *Web Service* is the chosen integration technique for transfering blogs within this project, it is important to design the interfaces in the right granularity to avoid unnecessary performance overhead. Preferable the definition of the interfaces should be one of the first tasks, because this ensures a stable interaction between the WSP and its WSCs. Furthermore the service design should avoid deep nested XML tree structures. The `MobileGateway` incorporates the following *Service Interfaces* for mobile user authentications and for gathering user blogs on the server domain:

  1. `boolean logon(String deviceID)`
     Used for a device-specific server logon. The implementation of this interface delegates the service parameter to the `IdentityManagement` component.

  2. `boolean logout(String deviceID)`
     Used for a device-specific server logout. The implementation of this interface delegates the service parameter to the `IdentityManagement` component.

  3. `boolean post(long timestamp, String journeyName, String message, double longitude, double latitude, double elevation, String encodedPhoto)`
     Used as a gateway for mobile blogs. The implementation of the interface maps the blog attributes into a *Java Bean* in order to be manageable by the `BlogContentHandler` service components.

- **Identity Management**
  This component reads the user's credentials from the underlying HTTP stream in order to verify them with the entries in the database table `user` as well as with the given device ID. If access is granted, the mobile user is set as *Online* by registering the according user name in the database table `trackeduser`. On a logout operation it resets the according entry in this database table, whereby the mobile session is terminated. The database is accessed in a transparent way by using the data-centric `PersistenceManagement` service component.

- **Blog Content Handler**
  This component interprets and handles the content of the received blog object. In a first step it decodes the binary data and reconstructs the according photo as far as defined. After a successful decoding, the *Photo* is stored in the user's repository together with a down-scaled *Thumbnail Photo* for preview purposes on the web clients. Regardless of the original photo dimension, the preview versions are equal-sized for all photo-related blogs. Furthermore the component performs the according entries into the database table `photo`, including the determined photo *Dimension* and the *Path* to the photo on the HDD. In the second step the actual blog attributes are recorded in the database table `blog`, including a reference to the photo-specific data in cases of non-message blogs.

  The `BlogContentHandler` service component has to ensure that the *Set of Blog Attributes* as defined in the tables 5.4 and 5.5 are made available in the database tables. Those sets are essential for the content supply to the web clients as indicated in chapter *Prototype*

Analysis - Web Client Domain [4.3.3]. In all cases, accesses to the host's data sources are performed in a transparent way by using the data-centric `PersistenceManagement` service component.

Attribute	Values
ID	'ID of Blog'
Longitude	'Longitude Value'
Latitude	'Latitude Value'
Elevation	'Elevation Value'(m) | n/a

Table 5.4.: *Minimum Set of Attributes for a web-based Waypoint Representation*

Attribute	Values
ID	'ID of Blog'
Timestamp	'Timestamp of Blog Creation'
Date	'Date and Time of Blog Creation'
Journey Name	'Name of Journey'
Message	'Message' | NULL
Photo Path	'Path the Photo' | NULL
Photo Path Thumb	'Path the Photo Thumbnail' | NULL
Photo Width	'Width of Photo' | -1
Photo Height	'Height of Photo' | -1

Table 5.5.: *Full Set of Attributes for a web-based Blog Representation*

- **Persistence Management**
  This component is a layer above the system's *Data Sources*, which has to deal with the actual data source-specific low level commands in order to perform the desired operations, received from related service components. As far as not managed by the RDBMS itself, the `PersistenceManagement` also has to ensure unique database entries as well as unique photo paths by using the blog's timestamp as file name.

5.2. Inter-Domain Communication

Interaction models enable an observation of the *Temporal Behaviour* as well as the *Temporal Order* of local and remote operations between the application domains. In the following figures, time-critical processes are illustrated and discussed. The noted estimated durations for the integration of remote services must not be exceeded, because otherwise this would result in system usability problems. Optionally service-specific *Web Service Level Agreement (WSLA)* parameters like network metrics might be used to determined the service quality (see section *Service Availability* [3.4.6.5]). The utilization of non-XML-based payload protocols would also speed-up service interactions in the mobile domain (see section *Payload Communication Strategies* [3.4.3]).

A typcial time-critical scenario is a unidirectional near *Real-Time Communication* between the mobile- and the web client domain through concurrently established server sessions, whereby a specific *Blog X* is created, posted and immediately available for an exploration in

the cartographic interface. The process in figure 5.5 describes a successfully creation of a waypoint-related *Blog X* with a resolved location. The process has to deal with the determination of the location datum and the retrieval of the according location identifiers by including device- and web-based services. It is considered that all services are available during this procedure, including an active satellite link. The methods' specification and the according parameters as defined in figures 5.5 and 5.6 are abstract and do not directly relate to class methods.

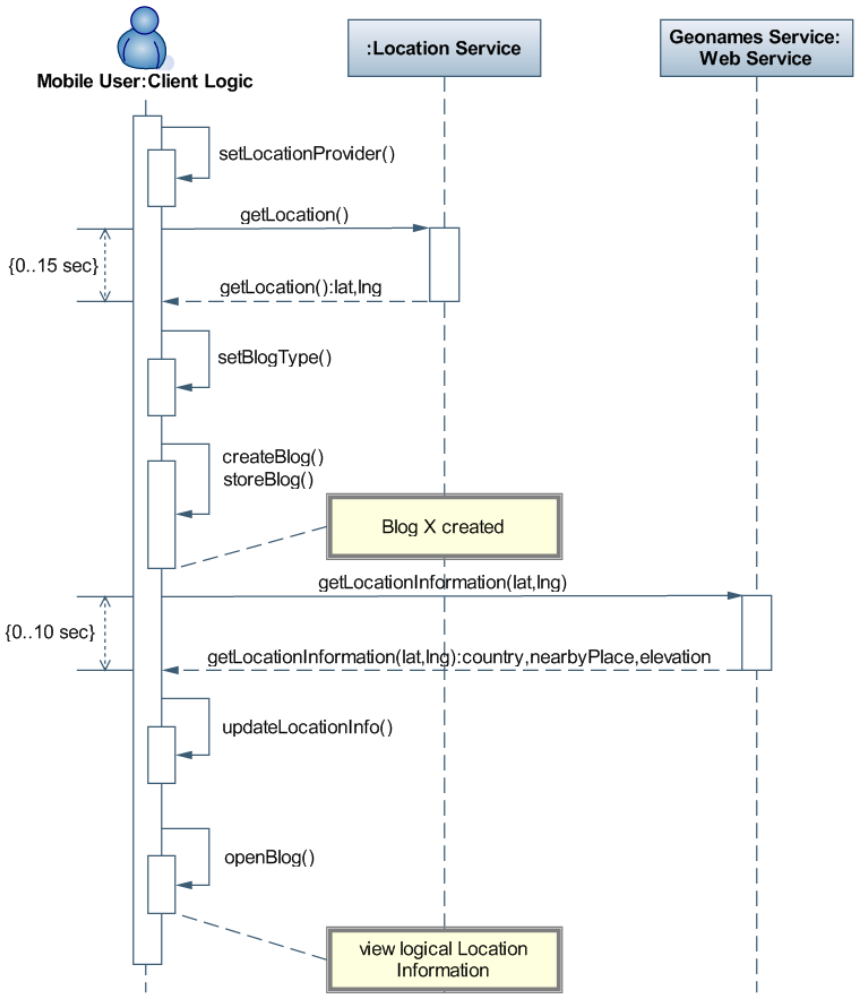


Figure 5.5.: *UML Sequence Diagram for Retrieving a Location Datum and its logical Location Information*

Figure 5.6 illustrates the interactions of the mobile- and the web client with the backend system in a concurrently way. After a successful user authentication has been performed on both clients, the user's waypoints and blogs are requested by the web client. As soon as the list response has been processed, further asynchronous data requests are scheduled with an interval of five seconds, in order to poll the server for updates. As a consequence, the subsequent posted and stored *Blog X* is considered by the server logic and directly delivered and integrated into the web client's GUI.

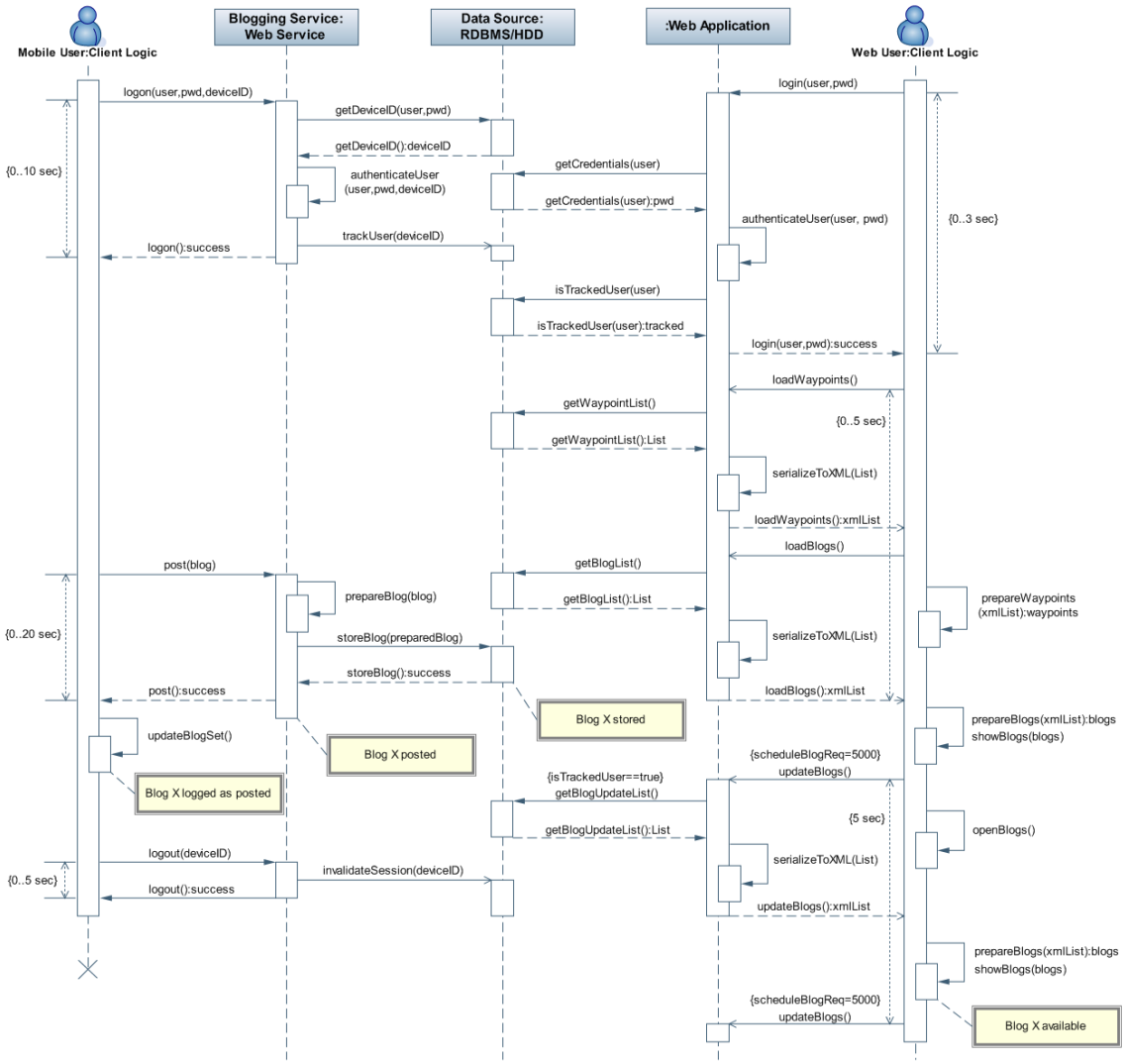


Figure 5.6.: *UML Sequence Diagram for Concurrently Mobile- and Web Client Sessions including the Server's Blog near Real-Time Updating Mechanism*

5.3. Deployment

The deployment diagram in figure 5.7 is used to define the actual application environments and their properties within the implemented components are executed. The diagram refers to the blogging application as *TMBlog*, which has been defined as the project's name.

The basic configuration of the *Web Application Execution Environment* as well as of the *Application-specific Parameters* is performed through the resources `server.xml` and `web.xml`, by default. These files specify the used communication *Ports*, *Security Realms* for authentication purposes, servlet *Paths*, and servlet-specific *Access Constraints*. The workflow, starting from an initial request up to the delivery of the response page, is managed through the `struts-config.xml` resource as part of the *Struts* framework. This configuration file is also used to define references to validators and to their underlying rules, as specified in the `validation.xml` and `validator-rules.xml` resources, as well as the data source properties including the used *JDBC* connector to the RDBMS. Since the *Web Service Execution Environ-*

ment is itself a web application in this context, it also needs a `web.xml` file. The actual *Web Service* only needs to hold its *WSDL* descriptor.

The basic MIDlet-based configuration files are the JavaME-specific *JAD* descriptor, which includes install- and execution instructions for the system's AMS, and the *J2ME Polish*-specific `messages.txt` file for the *Internationalization* and a suitable *Blog Content Configuration*.

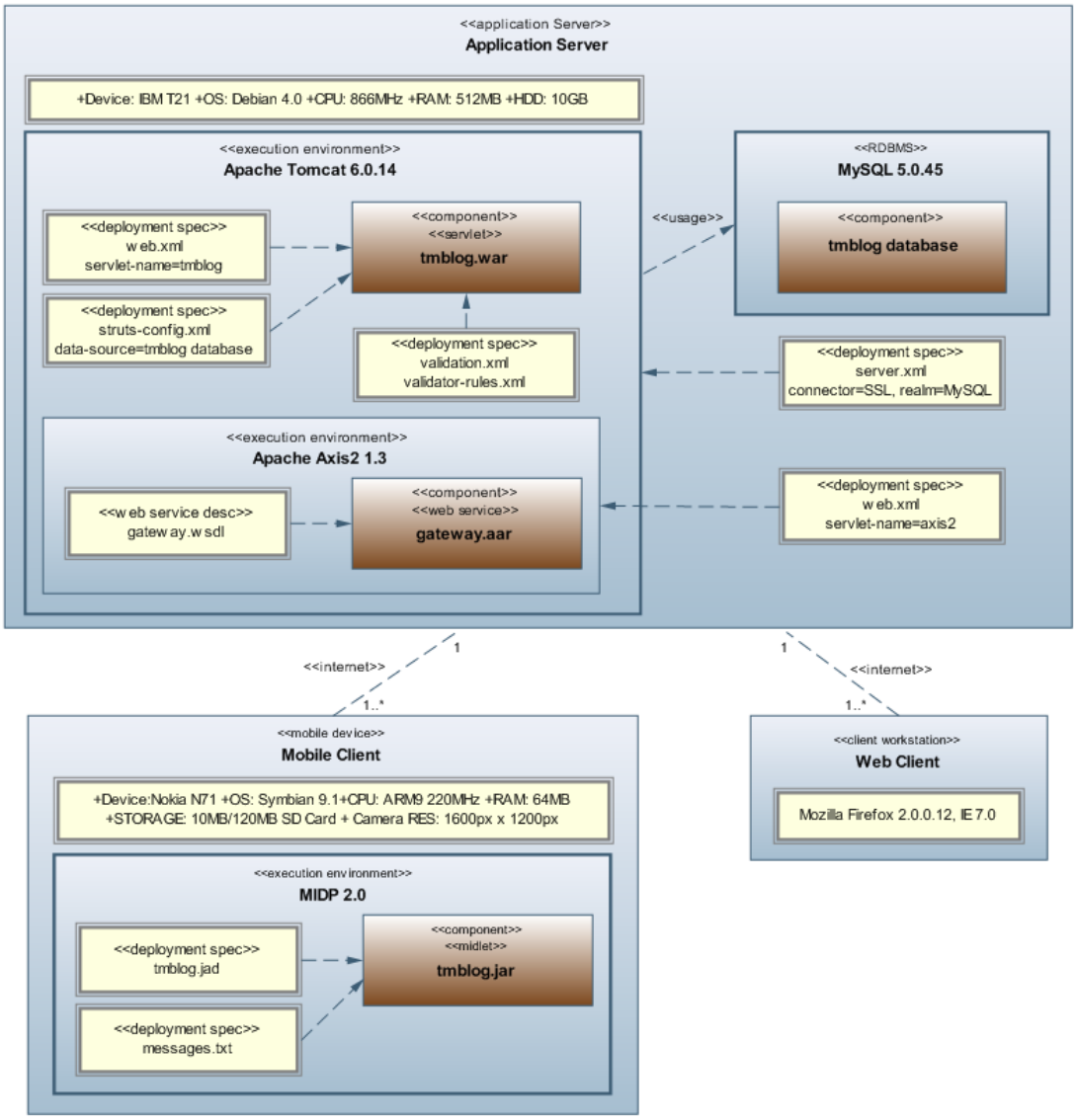


Figure 5.7.: *UML Deployment Diagram of the Blogging Application*

The widely deployed *Apache Tomcat* servlet container is used to execute the *Web Application* as well as the *Apache Axis2* SOAP engine, to enable the *Blogging Service* functionality. This combination seems to be suitable according to the dimension and the characteristics of this project. The *MySQL* database system has been chosen since it is reliable, easy to maintain, and free available as well.

6. Prototype Implementation

"The user does not know what he wants until he sees what he gets."

— Ed Yourdon
(Thoughts on Programming, Number 52)

In this chapter *Configuration Outlines*, *Class Diagrams*, and *Code Outlines* of the *Service Modules* as well as of the *Content Generation- and Access Management Subsystems* are discussed. The according *Listings* provide an insight of how distinct tasks have been realized. The *Prototype Implementation* refers to the three application domains, as they have been specified in the chapters *Requirements* [4] and *Prototype Design* [5].

6.1. Configuration

In order to offer the desired functionality of components within the chosen execution containers, a proper configuration of those and the application itself is essential. Among other things, this task enables the utilization of the container's built-in security features and a straightforward initiation of data sources, as well as an optimized building process. The captions of the following listings also refer to the resource in which the XML configuration code fragments are situated, as well as the execution environment and the according application domain.

Listing 6.1 shows three important sections of the *J2ME Polish*-specific *Ant Task* as part of the according *Application Building Process*, whereby some framework-specific instructions might override or extend IDE-specific ones. The *Info Section* defines some general information about the MIDlet, whereas the subsequent *Device Requirements Section* defines the devices for which the application should be optimized. Finally, the configuration within the *Build Process Section* controls the *Ant's* workflow.

```
<project name="TMBlog" default="j2mepolish">
 <target name="j2mepolish"
   depends="j2mepolish-init"
   description="This is the controller for the J2ME build process."
   >
  <j2mepolish>
   <!-- 1. info section -->
   <info copyright="Copyright 2007, 2008 Hannes Weingartner. All rights reserved."
    description="JavaME Application for the TMBlog Project."
    infoUrl="http://www.tmblog.org"
    jarName="${polish.vendor}-${polish.name}-${polish.locale}-TMBlogMobile.jar"
    jarUrl="${deploy-url}${polish.jarName}"
    icon="logo.png"
    name="TMBlogMobile"
    vendorName="TU Vienna"
    version="0.0.1" />

```

```
18    <!-- 2. device requirements section -->
19    <deviceRequirements unless="test">
20     <requirement name="JavaPlatform" value="MIDP/2.0+" />
21     <requirement name="Identifier" value="Nokia/N71"/>
22    </deviceRequirements>
23
24    <!-- 3. build process section -->
25    <build fullscreen="menu"
26        usePolishGui="yes"
27        workDir="${dir.work}"
28        destDir="${dir.dist}"
29        >
30     <!-- midlets definition -->
31     <midlets definition="${manifest.midlets}" if="manifest.midlets:defined" />
32     <midlets unless="manifest.midlets:defined">
33      <midlet class="at.ac.tuwien.control.Controller" name="TMBlog" />
34     </midlets>
35
36     <!-- project-wide variables - used for preprocessing  -->
37     <variables>
38      <variable file="configuration/configuration.properties" />
39     </variables>
40
41     <!-- customization settings -->
42     <resources
43      dir="resources"
44      defaultexcludes="yes"
45      excludes="readme.txt"
46      >
47      <root dir="resources/images" />
48      <root dir="resources/sounds" />
49
50      <!-- localization element for created localized versions of the application -->
51      <localization>
52       <locale name="en_US" unless="test" />
53       <locale name="en_US" if="test" />
54      </localization>
55     </resources>
56
57     <!-- logging framework settings -->
58     <debug showLogOnError="true"
59         verbose="true"
60         level="error"
61         if="test"
62         >
63      <filter pattern="at.ac.tuwien.control.controller" level="debug" />
64      <filter pattern="at.ac.tuwien.control.BlogManager" level="debug" />
65      <filter pattern="at.ac.tuwien.extern.webservice.tmblog.GatewayService" level="debug" />
66     </debug>
67     </build>
68   </j2mepolish>
69  </target>
70 </project>
```

Listing 6.1: `build.xml` - *MIDP: Mobile Client - J2ME Polish Ant Task*

Listing 6.2 defines the client- and database connections as part of the *Servlet Container* configuration, and listing 6.3 shows the web application-specific servlet definitions and -mappings as well as the security constraints for accessing them. The *Realm* for verifying valid web application users is defined as *MySQL*, whereby this code section further refers to the tables `user` and `userrole`.

```
<!-- http -->
<Connector  port="80"
   protocol="HTTP/1.1"
            connectionTimeout="20000"
            redirectPort="443" />

<!-- https -->
<Connector  port="443"
   protocol="HTTP/1.1" SSLEnabled="true"
            maxThreads="150" scheme="https" secure="true"
            clientAuth="false" sslProtocol="TLS" />

<!-- jdbc -->
<Realm className="org.apache.catalina.realm.JDBCRealm" debug="99"
            driverName="org.gjt.mm.mysql.Driver"
            connectionURL="jdbc:mysql://localhost/tmblog?user=tomcat&amp;password=ToMTMbloG"
            userTable="user"
            userNameCol="username"
            userCredCol="password"
            userRoleTable="userrole"
            roleNameCol="rolename"/>
```

Listing 6.2: `server.xml` - *Tomcat: Web Application - HTTP/HTTPS/JDBC Connection Configurations*

```
<!-- servlet settings -->
<servlet>
    <servlet-name>BlogLoader</servlet-name>
    <servlet-class>at.ac.tuwien.tmblog.web.actions.BlogLoader</servlet-class>
</servlet>
<servlet-mapping>
    <servlet-name>action</servlet-name>
    <url-pattern>*.do</url-pattern>
</servlet-mapping>

<servlet-mapping>
    <servlet-name>BlogLoader</servlet-name>
    <url-pattern>/BlogLoader</url-pattern>
    </servlet-mapping>
<session-config>
    <session-timeout>30</session-timeout>
    </session-config>
<welcome-file-list>
    <welcome-file>/jsp/index.jsp</welcome-file>
</welcome-file-list>

<!-- security settings -->
<security-constraint>
    <web-resource-collection>
        <web-resource-name>Protected Area</web-resource-name>
        <url-pattern>/*</url-pattern>
        <http-method>DELETE</http-method>
        <http-method>GET</http-method>
        <http-method>POST</http-method>
        <http-method>PUT</http-method>
    </web-resource-collection>
    <user-data-constraint>
        <description/>
        <transport-guarantee>CONFIDENTIAL</transport-guarantee>
    </user-data-constraint>
</security-constraint>
```

Listing 6.3: `web.xml` - *Tomcat: Web Application - Servlet and Security Configurations*

Listing 6.4 provides a collection of configuration sets for the *Struts* framework as part of the web application. Those settings include the *Action Form* definitions, the *Mappings* of the *JSPs* to the application logic, and the database connectivity configuration.

```
<struts-config>
 <!-- action form settings -->
    <form-beans>
        <form-bean name="StrutsActionLoginForm"
                   type="at.ac.tuwien.tmblog.web.forms.StrutsActionLoginForm"/>
            <form-property name="username" type="java.lang.String" />
            <form-property name="password" type="java.lang.String" />
        <form-bean name="StrutsActionRegisterForm"
                   type="at.ac.tuwien.tmblog.web.forms.StrutsActionRegisterForm">
            <form-property name="username" type="java.lang.String" initial=""/>
            <form-property name="password" type="java.lang.String" initial=""/>
            <form-property name="passwordConfirm" type="java.lang.String" initial=""/>
            <form-property name="deviceID" type="java.lang.String" initial=""/>
        </form-bean>
    </form-beans>

 <!-- smart links -->
    <global-forwards>
        <forward name="init" path="/init.do"/>
        <forward name="loginFwd" path="/loginFwd.do"/>
        <forward name="registerFwd" path="/registerFwd.do"/>
        <forward name="mainFwd" path="/mainFwd.do"/>
        <forward name="logoutFwd" path="/logoutFwd.do"/>
    </global-forwards>

 <!-- action mappings -->
    <action-mappings>
        <action path="/init"
                type="at.ac.tuwien.tmblog.web.actions.StrutsActionInit">
        </action>
        <action input="/jsp/login.jsp"
                name="StrutsActionLoginForm"
                attribute="StrutsActionLoginForm"
                path="/login"
                scope="session"
                validate="true"
                type="at.ac.tuwien.tmblog.web.actions.StrutsActionLogin">
        </action>
        <action input="/jsp/register.jsp"
                name="StrutsActionRegisterForm"
                attribute="StrutsActionRegisterForm"
                path="/register"
                scope="session"
                validate="true"
                type="at.ac.tuwien.tmblog.web.actions.StrutsActionRegister">
        </action>        <action path="/loginFwd"
                type="org.apache.struts.actions.ForwardAction"
                parameter="/jsp/login.jsp">
        </action>
        <action path="/registerFwd"
                type="org.apache.struts.actions.ForwardAction"
                parameter="/jsp/register.jsp">
        </action>
        <action path="/mainFwd"
                type="org.apache.struts.actions.ForwardAction"
                parameter="/jsp/main.jsp">
        </action>
        <action path="/logoutFwd"
                type="org.apache.struts.actions.ForwardAction"
                parameter="/jsp/logout.jsp">
        </action>
    </action-mappings>

 <!-- data source settings -->
    <data-sources>
        <data-source type="org.apache.tomcat.dbcp.dbcp.BasicDataSource">
            <set-property property="description" value="tmblog database" />
            <set-property property="driverClassName" value="com.mysql.jdbc.Driver" />
            <set-property property="url" value="jdbc:mysql://localhost:3306/tmblog" />
            <set-property property="username" value="tomcat" />
            <set-property property="password" value="ToMTMbloG" />
            <set-property property="maxActive" value="10" />
```

```
73              <set-property property="maxWait" value="5000" />
74              <set-property property="defaultAutoCommit" value="false" />
75              <set-property property="defaultReadOnly" value="false" />
76              <set-property property="validationQuery" value="SELECT COUNT(*) FROM user" />
77          </data-source>
78      </data-sources>
79
80      <message-resources parameter="at/ac/tuwien/tmblog/web/resources/application"/>
81
82  <!-- validator settings -->
83      <plug-in className="org.apache.struts.validator.ValidatorPlugIn">
84          <set-property
85              property="pathnames"
86              value="/WEB-INF/validator-rules.xml,/WEB-INF/validation.xml"/>
87      </plug-in>
88  </struts-config>
```

Listing 6.4: `struts-config.xml` - *Tomcat: Web Application - Struts Configuration*

Listing 6.5 refers to the security settings of the *Axis2 Web Application*, which are in fact those used for accessing the *Blogging Service* through mobile clients. The service container offers a transparent *HTTP Basic Authentication* for users of the *TMBlog* application.

```
1  <!-- security settings -->
2  <security-constraint>
3      <display-name>Web service Security</display-name>
4  <web-resource-collection>
5  <web-resource-name>Protected Area</web-resource-name>
6      <url-pattern>/services/Gateway</url-pattern>
7  </web-resource-collection>
8  <auth-constraint>
9      <role-name>tmblog</role-name>
10 </auth-constraint>
11 </security-constraint>
12 <security-role>
13     <role-name>tmblog</role-name>
14 </security-role>
15 <login-config>
16     <auth-method>BASIC</auth-method>
17     <realm-name>TMBlog Login</realm-name>
18 </login-config>
```

Listing 6.5: `web.xml` - *Axis2: Blogging Service - Security Configurations*

6.2. Service Modules

In the following, implementation issues of the distributed application service *Modules* are solved. A module consists of a *Service Producer-*, a *Service Consumer-*, and a *Service Connection* implementation. These modules belong to the *Mobile Client* domain, *Server- and Backend Systems* domain, and to the domains of independent service providers. Without any attention of third party service provider implementations, the module's *Class Diagrams* are illustrated and attended by outlines of the *Java Source Code* from included components, in the following section. With a focus on central functionality and due to clarity, the illustrated classes do not show all references to related components, as with the case of referenced objects in the source code. The captions of the following listings also refer to the resource in which the code fragments are situated, as well as the execution environment and the according application domain.

6.2.1. Location Service Module

Figure 6.1 illustrates the classes within the *Location Service Module* and their relationship among each other, with the `Location Manager` as the central service consumer component.

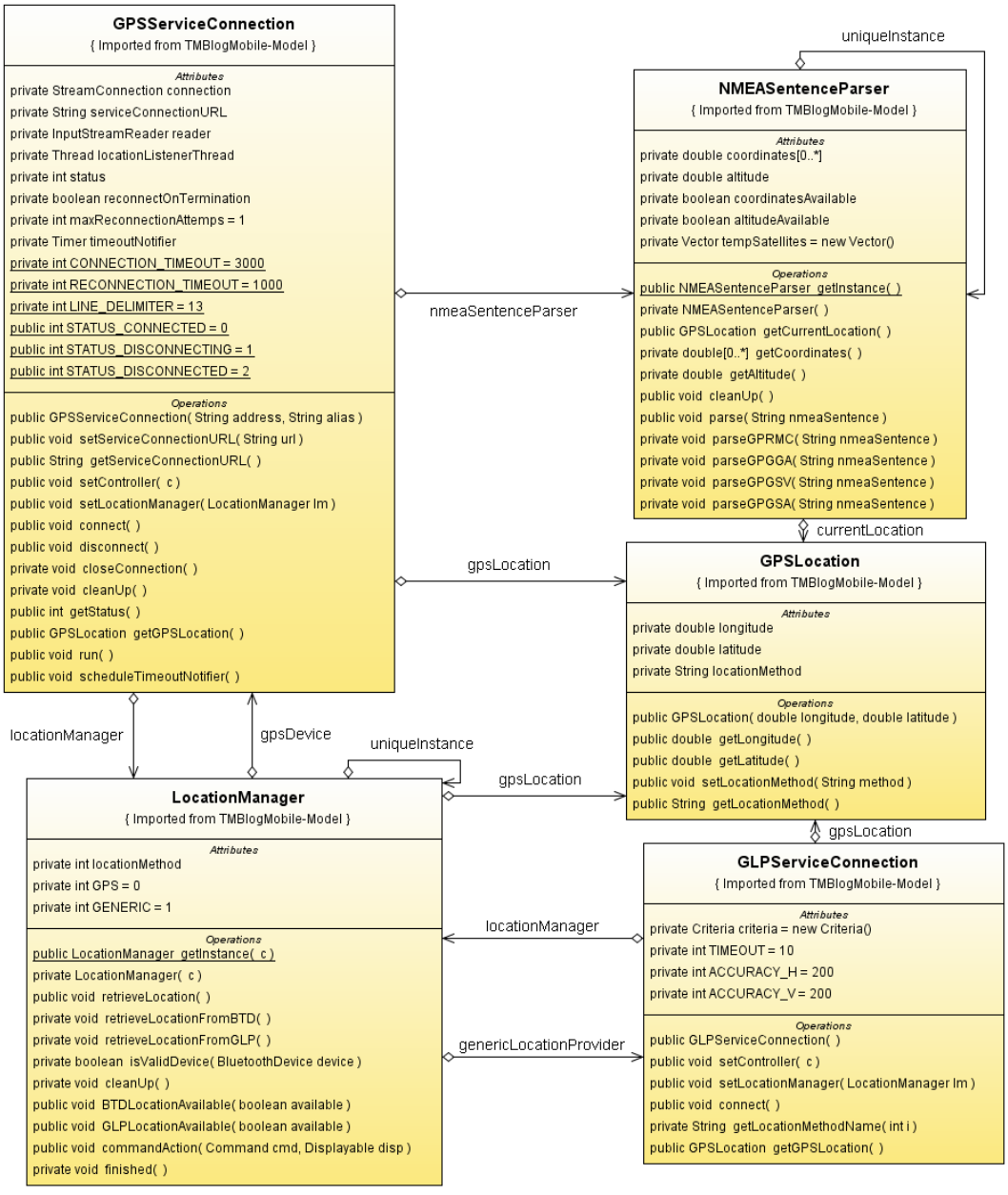


Figure 6.1.: *UML Class Diagram of Location Service Consumer- and Service Connection Components in the Mobile Client Domain*

The according listing 6.6 shows how a discovered GPS receiver is integrated into the application in order to retrieve the geodetic datum. Listing 6.7 utilizes the functionality of the *Location API* with the same aim. The according thread is started in the `retrieveLocationFromGLP()` method of the `Location Manager`.

```
private void retrieveLocationFromBTD() {
 this.locationMethod = this.GPS;
 String address = this.settings.getGPSDevice().getAddress();

 // check if GPS device has changed in the meantime
 if(this.gpsDevice == null || !this.gpsDevice.getAddress().equals(address)) {
    // create GPS device for receiving the GPS data from satellites
    String alias = this.settings.getGPSDevice().getAlias();
    this.gpsDevice = new GPSServiceConnection(address, alias);
    this.gpsDevice.setServiceConnectionURL(this.settings.getGPSDevice().getConnectionURL());
    this.gpsDevice.setController(this.controller);
    this.gpsDevice.setLocationManager(this);
 }
 if(this.gpsDevice.getStatus() == this.gpsDevice.STATUS_CONNECTED ||
    this.gpsDevice.getStatus() == this.gpsDevice.STATUS_DISCONNECTING) {
    return;
 }

 // GPS connection thread
 Thread t = new Thread() {
  int maxConnectionAttemps = 1;
  int connectionAttempts = 0;
  public void run() {
   while(connectionAttempts < maxConnectionAttemps
    && gpsDevice.getStatus() == gpsDevice.STATUS_DISCONNECTED) {
    try {
     connectionAttempts++;
     gpsDevice.connect();
     activityScreen.setString(Locale.get("locationManager.alert.waiting"));
    } catch(SecurityException se) {
     cleanUp();
     controller.show(controller.getScreen(controller.MAIN_SCR));
    } catch(IOException ioe) {
     if(connectionAttempts == this.maxConnectionAttemps) {
        cleanUp();
        Displayable nextScreen = controller.getScreen(controller.MAIN_SCR);
        controller.show(controller.ERROR_ASCR,
        Locale.get("locationManager.alert.unreachableDevice"), null, nextScreen);
     }
    }
   }
  }
 }; t.start();
}
```

Listing 6.6: `LocationManager.java` - *MIDP: Mobile Client - Connection Thread for a Location Retrival with a pre-selected BT-GPS Receiver*

```
// GLP connection thread
public void connect() {
    new Thread() {
        public void run() {
            try {
                LocationProvider provider = LocationProvider.getInstance(criteria);
                if(provider != null) {
                    try {
                        Location location = provider.getLocation(TIMEOUT);
                        //#debug
                        System.out.println("GLP - nmea string: "
                        +location.getExtraInfo("application/X-jsr179-location-nmea"));
                        if(location.isValid()) {
                            QualifiedCoordinates coord = location.getQualifiedCoordinates();
                            double lng = coord.getLongitude();
                            double lat = coord.getLatitude();
                            gpsLocation = new GPSLocation(lng, lat);
                            String method =
                            getLocationMethodName(location.getLocationMethod());
                            gpsLocation.setLocationMethod(method);
                            locationManager.GLPLocationAvailable(true);
                        }
                        if(gpsLocation == null)
```

```
24                              locationManager.GLPLocationAvailable(false);
25                          } catch(LocationException le) {
26                              locationManager.GLPLocationAvailable(false);
27                          } catch(InterruptedException ie) {
28                              locationManager.GLPLocationAvailable(false);
29                          } catch(SecurityException se) {
30                              controller.show(controller.getScreen(controller.MAIN_SCR));
31                          }
32                      } else
33                          throw new LocationException();
34                  } catch(LocationException le) {
35                      //#debug
36                      System.out.println("GLP: no Provider reachable.");
37                  }
38              }.start();
39      }
40  }
```

Listing 6.7: `GLPServiceConnection.java` - *MIDP: Mobile Client - Connection Thread for a Location Retrival with the Location API*

6.2.2. Location Resolution Service Module

Figure 6.2 illustrates the `GeonamesWSC` class, which is responsible for invoking the *Geonames* web service in a *RESTful* way. The resolved geodetic datum is stored in the `Toponym` class in order to be processed by further components.

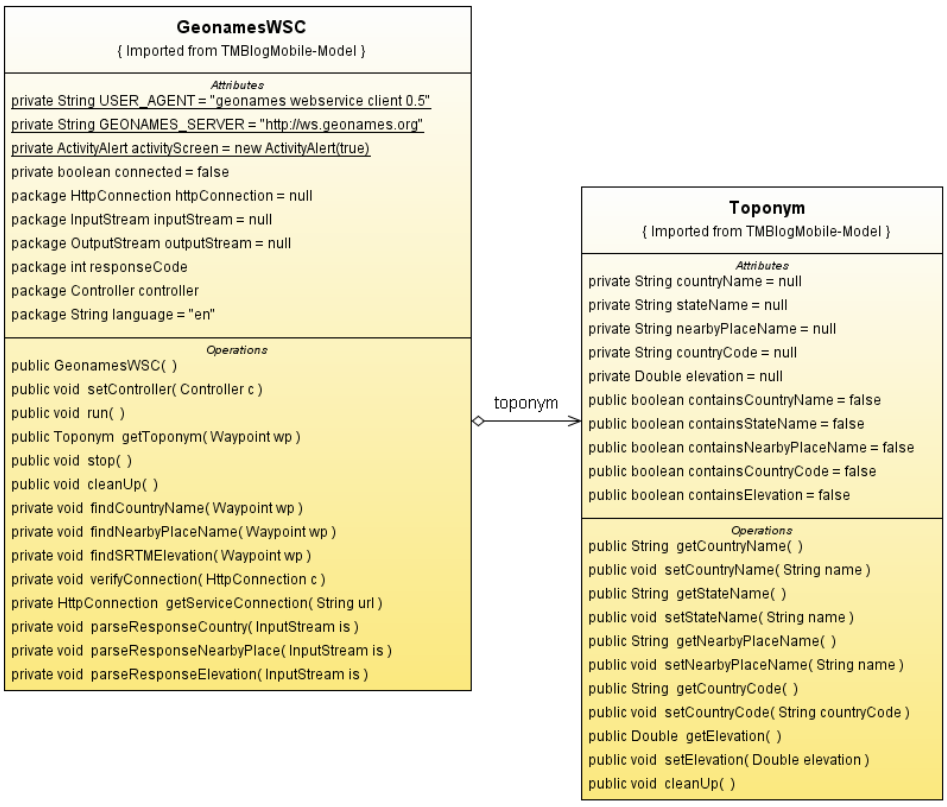


Figure 6.2.: *UML Class Diagram of the Location Resolution Service Consumer in the Mobile Client Domain*

Listing 6.8 shows how to determine the *Nearest Populated Place* for a specific waypoint. The system's *SAX Parser* is used to process the XML response from the service.

```
1  private void findNearbyPlaceName(Waypoint wp)
2   throws IOException, Exception, SecurityException {
3   // set request parameters
4   double latitude = wp.getLocation().getLatitude();
5   double longitude = wp.getLocation().getLongitude();
6
7   // service url
8   String url = GEONAMES_SERVER + "/findNearbyPlaceName?";
9   url += "type=XML";              // response type
10  url += "&style=SHORT";          // verbosity of returned xml document
11  url += "&lat=" + latitude;
12  url += "&lng=" + longitude;
13  url += "&lang" + this.language;
14  // open connection and send request
15  this.httpConnection = this.getServiceConnection(url);
16  // read the HTTP response headers
17  this.verifyConnection(this.httpConnection);
18  // get response stream
19  this.inputStream = this.httpConnection.openInputStream();
20  this.parseResponseNearbyPlace(this.inputStream);
21 }
22
23 // parse XML response to update toponym
24 private void parseResponseNearbyPlace(InputStream is) throws
25   FactoryConfigurationError,
26   ParserConfigurationException,
27   SAXException,
28   IOException {
29  SAXParser xmlParser = SAXParserFactory.newInstance().newSAXParser();
30  ResponseHandlerNearbyPlace handler = new ResponseHandlerNearbyPlace();
31  xmlParser.parse(is, handler);
32 }
```

Listing 6.8: `GeonamesWSC.java` - *MIDP: Mobile Client - Geonames Web Service Invocation for the Determination of the nearest populated Place*

Listing 6.9 shows instances of XML messages, as they have been received from the *Geonames* web service.

```
1  <!-- findNearbyPlace -->
2  <geonames>
3   <geoname>
4    <name>Ottakring</name>
5    <lat>48.2166667</lat>
6    <lng>16.3</lng>
7    <geonameId>2769359</geonameId>
8    <countryCode>AT</countryCode>
9    <countryName>Austria</countryName>
10   <fcl>P</fcl>
11   <fcode>PPLX</fcode>
12   <distance>2.3289</distance>
13  </geoname>
14 </geonames>
15
16 <!-- elevation -->
17 <geonames>
18  <srtm3>209</srtm3>
19  <lat>48.211225</lat>
20  <lng>16.3303533</lng>
21 </geonames>
```

Listing 6.9: *Instances of XML-Messages for the Transfer of resolved geodetic Data between the Geonames Web Service and the Mobile Clients*

6.2.3. Gateway Service Module

Figure 6.3 shows the `GatewayWSC` class with its high-level methods for accessing server-side functionality through the `GatewayServiceConnection` class. The `GatewayWSC` is being notified about remote events by the invocation of its *Callback Functions*, which are defined in the `IServiceListener` interface.

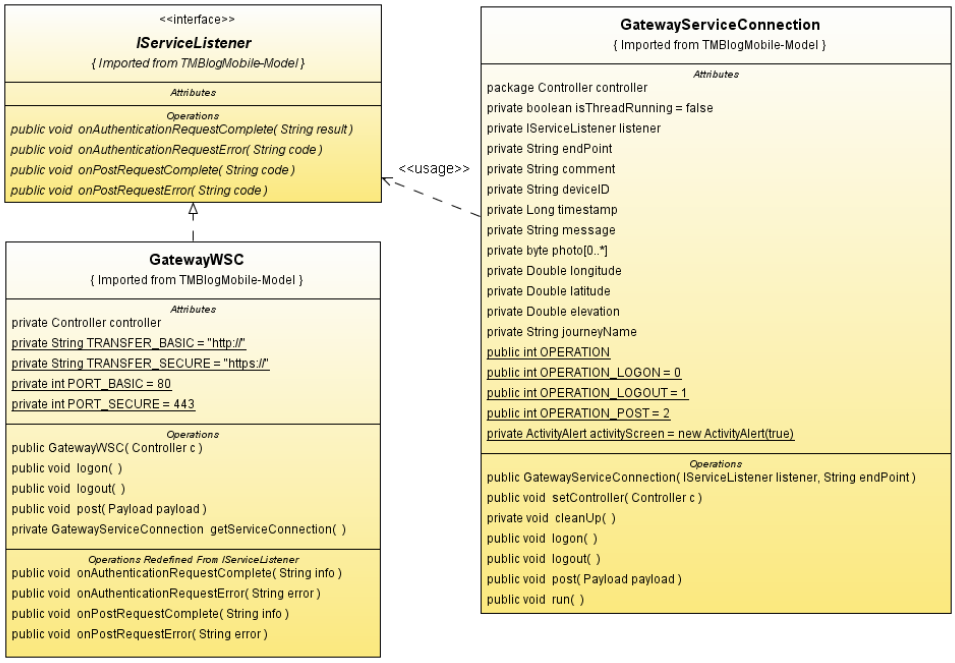


Figure 6.3.: *UML Class Diagram of Gateway Service Consumer- and Service Connection Components in the Mobile Client Domain*

Listing 6.10 shows how a remote user authentication is initiated by the *Mobile Client*. The user credentials are set as properties of the HTTP stream, which is managed through an instance of the service stub. Listing 6.11 illustrates how a mobile user authentication and blog reconstruction is handled through the server's *Blogging Service*.

```
1  // user authentication
2  public synchronized void logon() {
3   if (!isThreadRunning) {
4    isThreadRunning = true;
5    this.deviceID = this.controller.getCredentials().getBtAddress();
6    OPERATION = OPERATION_LOGON;
7    new Thread(this).start();
8   }
9  }
10
11 public void run() {
12  Gateway_Stub gatewayServiceStub = new Gateway_Stub();
13  gatewayServiceStub._setProperty(Stub.ENDPOINT_ADDRESS_PROPERTY, this.endPoint);
14  gatewayServiceStub._setProperty(Stub.SESSION_MAINTAIN_PROPERTY, new Boolean(false));
15
16  // credentials for user authentication on system (basic http authentication).
17  // username and password are base64 encoded by the JAX-RPC runtime before being send.
18  // since this an unreliable security policy an SSL tunnel is used additionally.
19  gatewayServiceStub._setProperty(
20   javax.xml.rpc.Stub.USERNAME_PROPERTY,
21   this.controller.getCredentials().getUser());
```

```
22  gatewayServiceStub._setProperty(
23   javax.xml.rpc.Stub.PASSWORD_PROPERTY,
24   this.controller.getCredentials().getKey());
25
26  // perform remote invocation
27  IGateway gatewayService = (IGateway) gatewayServiceStub;
28  if (OPERATION == OPERATION_LOGON) {
29   activityScreen.setString(Locale.get("gatewayServiceConnection.authenticate"));
30   this.controller.show(activityScreen);
31   try {
32       //#debug
33       System.out.println("invoke remote signon method...");
34    // pass the registered device ID as parameter
35    Boolean signed = gatewayService.signon(this.deviceID);
36    if (signed.booleanValue()) {
37     this.listener.onAuthenticationRequestComplete(
38     Locale.get("gatewayServiceConnection.accountAvailable"));
39    } else {
40     // user tries to login from a unregistered device
41     this.listener.onAuthenticationRequestError(
42     Locale.get("gatewayServiceConnection.noAccountAvailable"));
43    }
44   } catch (RemoteException re) {
45    // http authentication failed
46    // 401 unauthorized: invalid credentials
47    // 403 forbitten: no permissions
48    // 500 server error: invalid SOAP request or system-specific error
49    listener.onAuthenticationRequestError(
50    Locale.get("gatewayServiceConnection.serviceError"));
51   } catch (SecurityException se) {
52    this.controller.show(this.controller.getScreen(this.controller.LOGIN_SCR));
53   } catch (Exception e) {
54    // another IO problem e.g. Symbian Native Error-7372:
55    // an SSL tunnel can not be established - server is maybe down
56    listener.onAuthenticationRequestError(Locale.get("tmblogServer.serviceNotAvailable"));
57   }
58  }
59  this.cleanUp();
60 }
```

Listing 6.10: `GatewayServiceConnection.java` - *MIDP: Mobile Client - Connection Thread for the Remote User Authentication through the according Service Stub.*

```
1  // sign-on mobile user
2  public boolean signon(String deviceID) {
3   // get SOAP message context
4   MessageContext msgCxt = MessageContext.getCurrentMessageContext();
5   System.out.println("SOAP envelope: "+msgCxt.getEnvelope().toString());
6   // get username
7   HttpServletRequest request =
8   (HttpServletRequest)msgCxt.getProperty(HTTPConstants.MC_HTTP_SERVLETREQUEST);
9   String username = request.getRemoteUser();
10  boolean isValid = this.dbManager.isValidUser(username, deviceID);
11  // set user online
12  if(isValid) {
13      this.dbManager.trackUser(username);
14  }
15  return isValid;
16 }
17 // post blog
18 public boolean post (
19   long timestamp,
20   String journeyName,
21   String message,
22   double longitude,
23   double latitude,
24   double elevation,
25   String encodedPhoto) throws RemoteException {
26
27  MessageContext msgCxt = MessageContext.getCurrentMessageContext();
28  HttpServletRequest request =
```

```
29   (HttpServletRequest)msgCxt.getProperty(HTTPConstants.MC_HTTP_SERVLETREQUEST);
30  String username = request.getRemoteUser();
31  // check if user is online
32  if(!this.dbManager.isTrackedUser(username)) {
33   return false;
34  }
35  // create new blog
36  Blog blog = new Blog();
37  blog.setTimestamp(timestamp);
38  blog.setUserName(username);
39  blog.setJourneyName(journeyName);
40  blog.setMessage(message);
41  blog.setLongitude(longitude);
42  blog.setLatitude(latitude);
43  blog.setElevation(elevation);
44  // decode photo
45  byte[] photo = null;
46  if(encodedPhoto != null) {
47   photo = Base64Decoder.decode(encodedPhoto);
48   blog.setPhotoLength(photo.length);
49   blog.setPhoto(photo);
50  } else {blog.setPhotoLength(0);}
51  try {
52   if(blog.isValid()) {
53    this.dbManager.storeBlog(blog);
54   } else {throw new RemoteException();}
55  } catch (SQLException se) {
56   throw new RemoteException();
57  } catch (IOException ioe) {
58   throw new RemoteException();
59  }
60  System.out.println("blog stored.");
61  return true;
62 }
```

Listing 6.11: `Gateway.java` - *Axis2: Blogging Service - Mobile User Authentication and Blog Reconstruction*

The code for the `IGateway` interface and the according `Gateway_Stub` is generated by the IDE, out of the *Gateway Service WSDL Description*. The input type- and service definitions of this contract are offered in listing 6.12.

```
1  <!-- wsdl type definitions -->
2  <wsdl:types>
3   <xs:schema
4    xmlns:ns="http://webservice.tmblog.tuwien.ac.at"
5    attributeFormDefault="qualified"
6    elementFormDefault="qualified"
7    targetNamespace="http://webservice.tmblog.tuwien.ac.at">
8    <xs:element name="signon">
9        <xs:complexType>
10     <xs:sequence>
11               <xs:element minOccurs="0" name="deviceID" nillable="true" type="xs:string"/>
12           </xs:sequence>
13       </xs:complexType>
14   </xs:element>
15   <xs:element name="post">
16       <xs:complexType>
17           <xs:sequence>
18               <xs:element minOccurs="0" name="timestamp" type="xs:long"/>
19               <xs:element minOccurs="0" name="journeyName" nillable="true" type="xs:string"/>
20               <xs:element minOccurs="0" name="message" nillable="true" type="xs:string"/>
21               <xs:element minOccurs="0" name="longitude" type="xs:double"/>
22               <xs:element minOccurs="0" name="latitude" type="xs:double"/>
23               <xs:element minOccurs="0" name="elevation" type="xs:double"/>
24               <xs:element minOccurs="0" name="encodedPhoto" nillable="true" type="xs:string"/>
25           </xs:sequence>
26       </xs:complexType>
```

```
27   </xs:element>
28   <xs:element name="logout">
29       <xs:complexType>
30           <xs:sequence>
31               <xs:element minOccurs="0" name="deviceID" nillable="true" type="xs:string"/>
32           </xs:sequence>
33       </xs:complexType>
34   </xs:element>
35  </xs:schema>
36 </wsdl:types>
37
38 <!-- wsdl service definition -->
39 <wsdl:service name="Gateway">
40     <wsdl:port name="GatewaySOAP11port_http" binding="ns2:GatewaySOAP11Binding">
41         <soap:address location="http://192.168.1.35:80/axis2/services/Gateway"/>
42     </wsdl:port>
43     <wsdl:port name="GatewaySOAP12port_http" binding="ns2:GatewaySOAP12Binding">
44         <soap12:address location="http://192.168.1.35:80/axis2/services/Gateway"/>
45     </wsdl:port>
46     <wsdl:port name="GatewayHttpport" binding="ns2:GatewayHttpBinding">
47         <http:address location="http://192.168.1.35:80/axis2/services/Gateway"/>
48     </wsdl:port>
49 </wsdl:service>
```

Listing 6.12: `wsdl.xml` - *Axis2: Blogging Service* - WSDL Type and Service Definition

Listing 6.13 shows two instances of *SOAP Envelopes* as they are transfered to the server. The first message represents a *Message Blog*, the second one a *Photo Blog*.

```
1  <?xml version='1.0' encoding='utf-8'?>
2  <soap:Envelope xmlns:soap="http://schemas.xmlsoap.org/soap/envelope/"
3        xmlns:tns="http://webservice.tmblog.tuwien.ac.at"
4        xmlns:xsi="http://www.w3.org/2001/XMLSchema-instance"
5        xmlns:xsd="http://www.w3.org/2001/XMLSchema">
6   <soap:Body>
7    <tns:post>
8     <tns:timestamp>1205393379066</tns:timestamp>
9     <tns:journeyName>My Journey</tns:journeyName>
10    <tns:message>friedmanngasse</tns:message>
11    <tns:longitude>16.3303533</tns:longitude>
12    <tns:latitude>48.211225</tns:latitude>
13    <tns:elevation>209.0</tns:elevation>
14    <tns:encodedPhoto xsi:nil="true" />
15   </tns:post>
16  </soap:Body>
17 </soap:Envelope>
18
19 <?xml version='1.0' encoding='utf-8'?>
20 <soap:Envelope xmlns:soap="http://schemas.xmlsoap.org/soap/envelope/"
21       xmlns:tns="http://webservice.tmblog.tuwien.ac.at"
22       xmlns:xsi="http://www.w3.org/2001/XMLSchema-instance"
23       xmlns:xsd="http://www.w3.org/2001/XMLSchema">
24  <soap:Body>
25   <tns:post>
26    <tns:timestamp>1205319419217</tns:timestamp>
27    <tns:journeyName>My Journey</tns:journeyName>
28    <tns:message>karlskirche</tns:message>
29    <tns:longitude>16.371295</tns:longitude>
30    <tns:latitude>48.1987233</tns:latitude>
31    <tns:elevation>187.0</tns:elevation>
32    <tns:encodedPhoto>/9j/4AAQSkZJRgABAQAAAQABAAD/2wCEA...</tns:encodedPhoto>
33   </tns:post>
34  </soap:Body>
35 </soap:Envelope>
```

Listing 6.13: *Instances of the SOAP Envelopes for the Blog Transfer between Mobile Clients and the Blogging Service*

Figure 6.4 shows the generated *Service Stub* at the according *Interface*. The invocation of the stub's service methods causes a marshalling of the passed parameters.

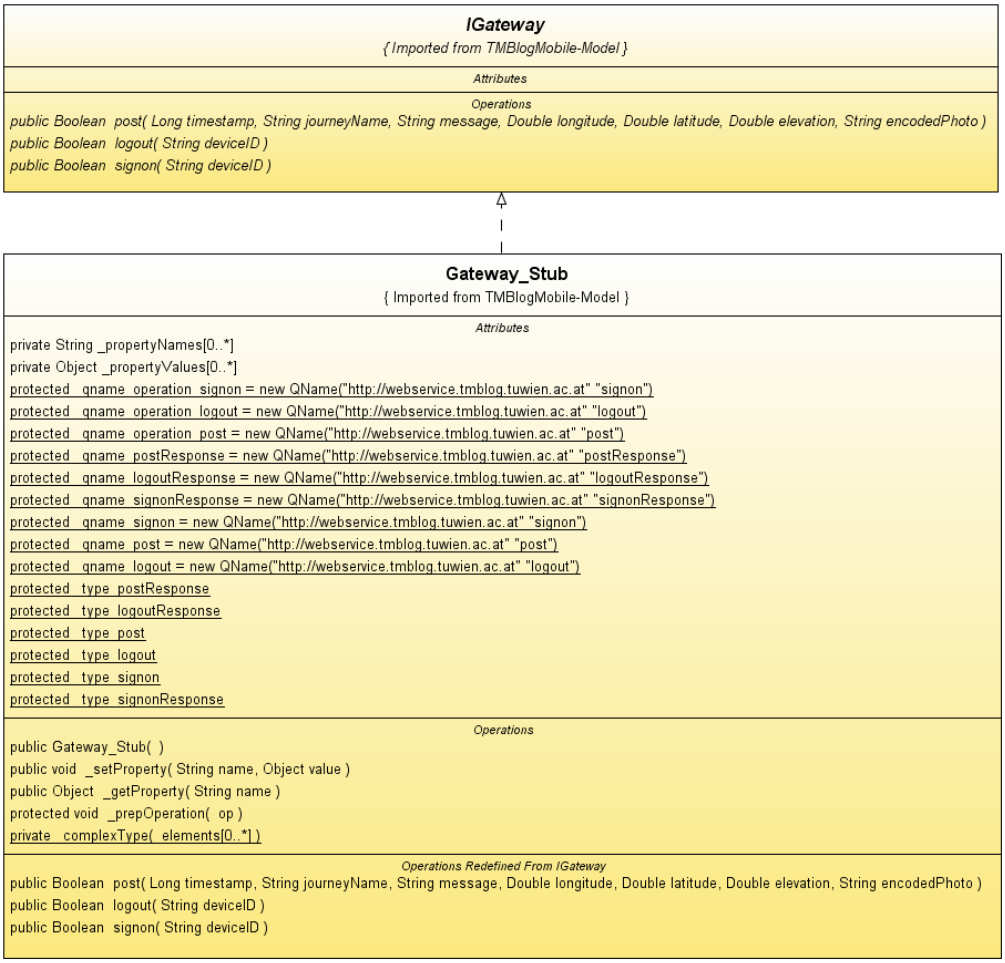


Figure 6.4.: *UML Class Diagram of Gateway Service Connection Components in the Mobile Client Domain - Service Interface and Service Stub*

6.3. Content Generation Management

The blog generation process is managed by the `BlogManager` and the `PhotoManager` classes. The `PhotoManager` contains the inner class `PhotoFormatter`, which is responsible for mapping the photo's content to a *Photo* object, including a *Thumbnail* version. The formatter is also used during the capture process to offer the user a *Preview* of the blog's content. The dimension and the encoding for the photo is parsed from the application's configuration file. In order to create message blogs, only a text input mask is required. Figure 6.5 shows the class diagram of the three mentioned components as well as their relationship among each other.

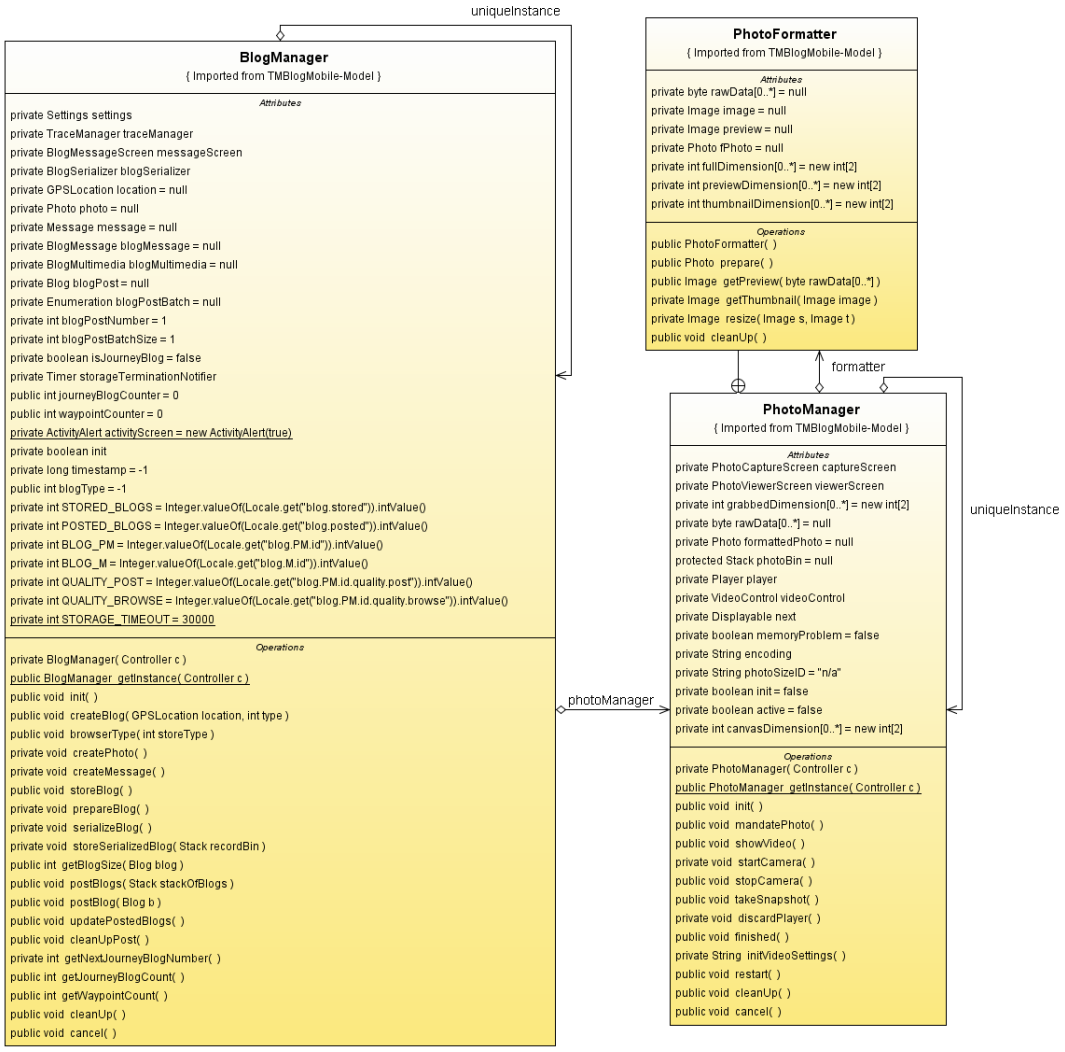


Figure 6.5.: *UML Class Diagram of Content Generation Management Components in the Mobile Client Domain*

Listing 6.14 shows the methods for initializing the blog content generation process. During the photo capture phase, the `BlogManager` is suspended until the finished `Photo` object has been passed to the manager within the `PhotoBin` stack. Listing 6.15 illustrates the tasks performed within the `takeSnapshot()` method, called from the application in order to process the buffered video still.

```
1  // photo
2  private void createPhoto() {
3   // init photo capture process
4   this.photoManager.mandatePhoto();
5   // photo waiting thread
6   Thread t = new Thread() {
7
8    public void run() {
9     BlogManager m = BlogManager.uniqueInstance;
10    Stack pb = m.photoManager.photoBin; // lock
11    m.photo = null;
12    synchronized (pb) {
13     try {
14      pb.wait();
15     } catch (InterruptedException ie) {/*ignore*/
```

```
16      }
17      // look for a photo
18      if (!pb.isEmpty()) {
19       m.photo = (Photo) pb.pop();
20       m.createMessage();
21      }
22     }
23    }
24   };
25   t.start();
26  }
27
28  // message
29  private void createMessage() {
30   this.messageScreen.setString(null);
31   this.controller.show(this.messageScreen);
32  }
```

Listing 6.14: `BlogManager.java` - *MIDP: Mobile Client - Operations for the Initialization of the Blog Content Generation Process*

```
1  public void takeSnapshot() {
2   this.captureScreen.setStatus(PhotoCaptureScreen.STATUS_INITIALIZING);
3   if(this.player != null) {
4    try{
5     // get raw jpeg photo data
6     this.rawData = videoControl.getSnapshot(this.encoding);
7     if(this.rawData != null) {
8      Image previewPhoto = null;
9      previewPhoto = this.formatter.getPreview(this.rawData);
10     // delete last preview
11     this.viewerScreen.deleteAll();
12     //#style photoViewerTitleItem
13     this.viewerScreen.append(Locale.get("photoManager.label.preview")+": ");
14     // set new preview
15     //#style photoViewerPhotoItem
16     this.viewerScreen.append(previewPhoto);
17     // set snapshot info
18     //#style photoViewerInfoItem
19     this.viewerScreen.append(Locale.get("photoManager.label.photo.size")+": "
20     +this.photoSizeID+" ("+this.grabbedDimension[0]+" x "+this.grabbedDimension[1]+")");
21     // show preview screen
22     this.controller.show(this.viewerScreen);
23    }
24   } catch(MediaException me) {
25    //#debug error
26    System.out.println("Capture - Media Exception: "+me);
27    this.memoryProblem = true;
28    this.cleanUp();
29    this.controller.show(this.controller.ERROR_ASCR,
30    Locale.get("photoManager.alert.captureFailed")
31    +Locale.get("photoManager.alert.memoryProblem"), null, this.next);
32   } catch(SecurityException se) {
33    this.cleanUp();
34    this.controller.show(this.controller.getScreen(this.controller.BLOG_SCR));
35   } catch(Exception e) {
36    this.controller.show(this.controller.ERROR_ASCR,
37    Locale.get("photoManager.alert.captureFailed")
38    +Locale.get("photoManager.alert.memoryProblem"), null, this.next);
39   }
40  }
41  this.captureScreen.setStatus(PhotoCaptureScreen.STATUS_READY);
42 }
```

Listing 6.15: `PhotoManager.java` - *MIDP: Mobile Client - Operations on the buffered Video Still during the Photo Capture Process*

6.4. Content Access Management

The *Content Access Management* subsystem, as illustrated in figure 5.3, contains the components *Session Listener* and *Blog Loader*. The *Session Listener* is manifested as a `SessionBindingListener` instance, created by the `StrutsActionLogin` class, in cases where a web client has been authenticated successfully. The component's functionality is bound on the behaviour of the user's session status, and used to perform waypoint and blog database queries in order of the `BlogLoader` class. Figure 6.6 shows the relationship between these web application components.

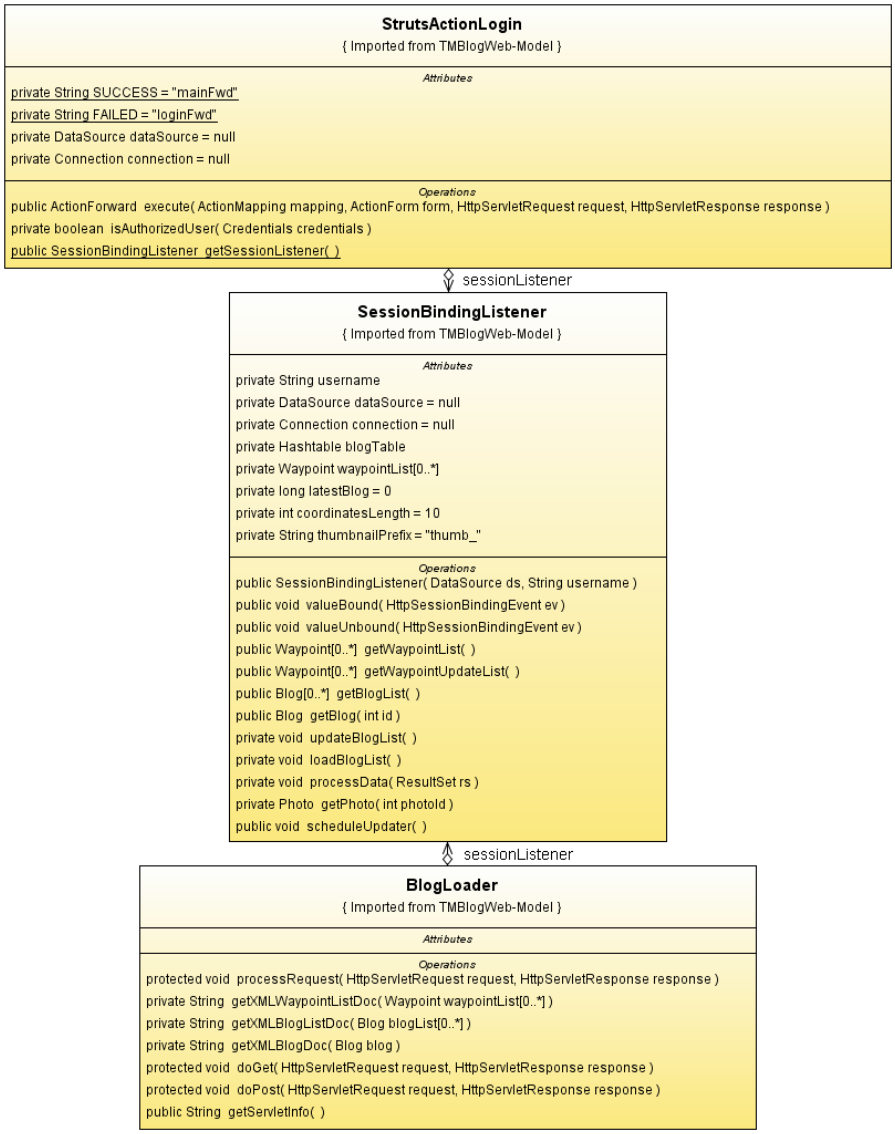


Figure 6.6.: *UML Class Diagram of Content Access Management Components in the Server- and Backend Systems Domain*

Listing 6.16 refers to the *Web Client Authentication* functionality, which includes the application's *Session Management*. According to the session status, either the `valueBound()` or the `valueUnbound()` method of the `SessionBindingListener` instance is called. As a

consequence, the listener either establishes or closes a connection to the user's data sources on occured `HttpSessionBindingEvents`.

```
// web client authentication
public ActionForward execute(ActionMapping mapping, ActionForm form,
  HttpServletRequest request, HttpServletResponse response)
  throws Exception {

 StrutsActionLoginForm loginForm = (StrutsActionLoginForm) form;

 ActionMessages errors = new ActionErrors();
 Credentials credentials = new Credentials();
 credentials.setUsername(loginForm.getUsername());
 credentials.setPassword(loginForm.getPassword());
 sessionListener = null;

 try {
  this.dataSource = getDataSource(request);
  this.connection = dataSource.getConnection();

  // db interaction
  if (this.isAuthorizedUser(credentials)) {
   System.out.println("user: " + credentials.getUsername() + " is authenticated.");
   // get session
   HttpSession session = request.getSession();
   session.setAttribute("login.done", credentials.getUsername());
   // maintain inactive session for half an hour
   session.setMaxInactiveInterval(1800);
   // add a session binding listener
   sessionListener =  new SessionBindingListener(this.dataSource, credentials.getUsername());
   session.setAttribute("bindings.listener", sessionListener);
   return mapping.findForward(SUCCESS);
  } else {
   errors.add(ActionMessages.GLOBAL_MESSAGE, new ActionMessage("errors.login.failed"));
   System.out.println("user: " + credentials.getUsername() + " is not authorized.");
  }
 } catch (SQLException sqle) {
  errors.add(ActionMessages.GLOBAL_MESSAGE, new ActionMessage("errors.db.failed"));
  getServlet().log("Connection.process", sqle);
 } finally {
  try {
   connection.close();
  } catch (SQLException e) {
   getServlet().log("Connection.close", e);
  }
 }
 // save error in request
 saveErrors(request, errors);
 return mapping.findForward(FAILED);
}
```

Listing 6.16: `StrutsActionLogin.java` - *Tomcat: Web Application - Session Binding through the Application's Web Client Authentication Functionality*

Listing 6.17 refers to the `BlogLoader` class, which represents a *Servlet* for processing incoming asynchronous waypoint and blog requests. The source code of the component also shows how to utilize the *XStream* tool to perform the mapping of the corresponding *Java Bean* into a *XML Representation*.

```
protected void processRequest(HttpServletRequest request, HttpServletResponse response)
  throws ServletException, IOException {

 String action = request.getParameter("action");
 response.setContentType("application/xml");
 ServletOutputStream out = response.getOutputStream();
 this.sessionListener = StrutsActionLogin.getSessionListener();
```

```
8  try {
9   if (action != null && action.equals("getWaypointList")) {
10   // create xml document and send it to client
11   out.println(this.getXMLWaypointListDoc(this.sessionListener.getWaypointList()));
12   out.flush();
13  } else if (action != null && action.equals("getWaypointUpdateList")) {
14   out.println(this.getXMLWaypointListDoc(this.sessionListener.getWaypointUpdateList()));
15   out.flush();
16  } else if (action != null && action.equals("getBlogList")) {
17   // create xml document and send it to client
18   out.println(this.getXMLBlogListDoc(this.sessionListener.getBlogList()));
19   out.flush();
20  } else if (action != null && action.equals("getBlog")) {
21   String id = request.getParameter("id");
22   // create xml document and send it to client
23   if (id != null) {
24    out.println(
25    this.getXMLBlogDoc(this.sessionListener.getBlog(Integer.valueOf(id).intValue())));
26    out.flush();
27   }
28  }
29 } finally {
30  out.close();
31 }
32 }
33
34 private String getXMLWaypointListDoc(Vector<Waypoint> waypointList) {
35  StringBuffer xmlString = new StringBuffer();
36
37  xmlString.append("<list>");
38  if (waypointList != null) {
39   XStream xstream = new XStream();
40   for (int i = 0; i < waypointList.size(); i++) {
41    xstream.alias("waypoint", Waypoint.class);
42    String xml = xstream.toXML(waypointList.elementAt(i));
43    xmlString.append(xml);
44   }
45  }
46  xmlString.append("</list>");
47  System.out.println("waypoint xml string: " + xmlString.toString());
48  return xmlString.toString();
49 }
```

Listing 6.17: `BlogLoader.java` - *Tomcat: Web Application - Servlet for processing Waypoint- and Blog Requests from Web Clients*

The *Java Script* code in listing 6.18 outlines how the `WaypointManager` class as part of the *Web Client Logic* performs the mentioned asynchronous requests to the `BlogLoader` in order to retrieve the waypoints of the posted blogs. A registered *Callback Function* is invoked as soon as the server replies with a HTTP status code. A valid XML response is parsed by reading the according elements within the DOM tree. If at least one waypoint has been loaded, a *Marker/Pin* for the *Map Interface* is going to be created and set to the corresponding location. Finally, the web client's `BlogManager` class requests the server for the actual blogs, which include all remaining non-location related blog attributes.

```
function loadWaypointList() {
 if(map != null) {
  // create browser-safe XMLHttpRequest object
  var request = GXmlHttp.create();
  // Prepare an asynchronous HTTP request to the server
  request.open("GET", "/tmblog/BlogLoader?action=getWaypointList", true);
  // Returned data will be processed by this processWaypoints
  request.onreadystatechange = getCallbackFunction(request, processXMLResponse);
  request.send(null);
  isLiveUpdate = false;
 }
};
function getCallbackFunction(request, processXMLResponse) {
 return function () {
  // request complete and ok
  if (request.readyState == 4) {
   if (request.status == 200) {
    // Pass the XML payload of the response to the handler function
    processXMLResponse(request.responseXML);
   } else {
   // http error
   alert("HTTP error: "+request.status);
   }
  }
 }
};
function processXMLResponse(waypointsXML){
    // obtain the array of waypoints and loop through it
    var waypointList = waypointsXML.documentElement.getElementsByTagName("waypoint");
    for (var i = 0; i < waypointList.length; i++) {
        // obtain the attributes of each marker
        var id = waypointList[i].getElementsByTagName("id")[0].firstChild.nodeValue;
        var lng = parseFloat(
 waypointList[i].getElementsByTagName("longitude")[0].firstChild.nodeValue);
        var lat = parseFloat(
 waypointList[i].getElementsByTagName("latitude")[0].firstChild.nodeValue);
        var ele = parseFloat(
 waypointList[i].getElementsByTagName("elevation")[0].firstChild.nodeValue);
        // create a new blog to be displayed on demand
        var blog = new Blog();
        blog.setID(id);
        blog.setLongitude(lng);
        blog.setLatitude(lat);
        blog.setElevation(ele);
        // store new blog
        blogManager.setBlog(blog);
        // create a display a new waypoint for the map
        var params = new Array();
        params[0] = id;
        params[1] = lat;
        params[2] = lng;
        createWaypoint(params);
    }
    if(waypointList.length > 0) {
        blogManager.loadBlogList();
    }
    // schedule next server poll
    setTimeout("waypointManager.loadWaypointUpdateList();", pollInterval);
};
```

Listing 6.18: `WaypointManager.js` - *Requesting and Parsing Waypoints in the Web Client Domain*

Listings 6.19 and 6.20 show two instances of Waypoints and Blogs, as they are transfered from the Web Application to the Web Client. The *Blog ID* attribute is used to request specific blogs from the server as well as for client-side content management purposes. After the initial blog retrieval process, an empty list or a list with newly created blogs is delivered in predefined request interval, which is five seconds in the prototype implementation.

```
<list>
 <waypoint>
   <id>1</id>
   <longitude>16.3303533</longitude>
   <latitude>48.211225</latitude>
   <elevation>209.0</elevation>
 </waypoint>
 <waypoint>
   <id>2</id>
   <longitude>16.371295</longitude>
   <latitude>48.1987233</latitude>
   <elevation>187.0</elevation>
 </waypoint>
</list>
```

Listing 6.19: *Instances of XML-Messages for the Transfer of Waypoints between the Web Application and the Web Client*

```
<list>
 <blog>
   <id>1</id>
   <timestamp>1205393379066</timestamp>
   <date>2008/03/13 08:29:39</date>
   <journeyName>My Journey</journeyName>
   <message>friedmanngasse</message>
   <photoPath>NULL</photoPath>
   <photoPathThumb>NULL</photoPathThumb>
   <photoWidth>-1</photoWidth>
   <photoHeight>-1</photoHeight>
   <onDemand>0</onDemand>
 </blog>
 <blog>
   <id>2</id>
   <timestamp>1205319419217</timestamp>
   <date>2008/03/12 11:56:59</date>
   <journeyName>My Journey</journeyName>
   <message>karlskirche</message>
   <photoPath>/repository/hannes/photos/1205319419217.jpg</photoPath>
   <photoPathThumb>/repository/hannes/photos/thumb_1205319419217.jpg</photoPathThumb>
   <photoWidth>320</photoWidth>
   <photoHeight>240</photoHeight>
   <onDemand>0</onDemand>
 </blog>
</list>
```

Listing 6.20: *Instances of XML-Messages for the Transfer of Blogs between the Web Application and the Web Client*

6.5. Interface

This section depicts *Screenshots* of the *Mobile*- and the *Web Client*. In both domains *CSS* files have been used to structure and to define the screen's content. In the same way, GUI messages are outsourced to domain-specific text files. This allows an easy adaption of the application's *Look and Feel* and its *Locale*. The figure sets 6.7, 6.8, 6.9, 6.10, and 6.11 refers to different system states and user interactions in the mobile domain.

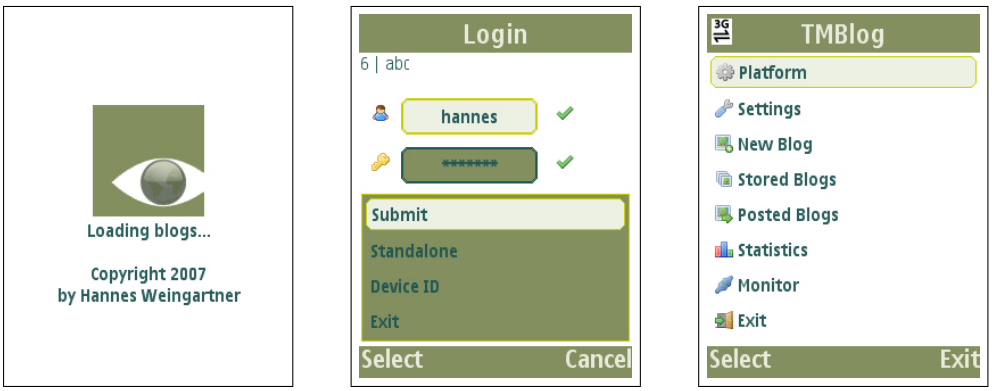


Figure 6.7.: *Screenshots of the Mobile Client - The Application Startup Process loads the Components as well as the stored Blogs (left). The Login Screen verifies the User Input before Credentials are allowed to be submitted (center). The Application's Main Menu is used to initiate Tasks (right).*

Figure 6.8.: *Screenshots of the Mobile Client - J2ME Polish specific Tabbed Forms for a user-friendly System Preference Browsing and Setting, categorized into different logical Domains.*

Figure 6.9.: *Screenshots of the Mobile Client - The System's Security-related Capability Feature confirms the User during the Location Determination Process since the Appliation is not Signed (left). Exploration of stored Waypoint-related Journey Blogs. The List Items represent generated Blogs and refer to their Content as well as to their spatial and temporal Relationship among each other (center). Several Operation can be performed on stored Blogs (right)*

Figure 6.10.: *Screenshots of the Mobile Client - J2ME Polish specific Tabbed Forms for a categorized Content-Browsing within an opened Photo Blog (left). Exploration of logical Location Information within an opened Blog (center). Journey- and Blog Statistics are used as a Tracking und Content Management Feature (right).*

Figure 6.12 shows the used Nokia handset and the GPS receiver. Figure 6.13, 6.14, and 6.15 refer to the cartographic web interface which is used for the exploration of the posted blogs as well as the user registration form. The map functionality relies on the implemention of the *Google Maps* web service and an optional package for the definition of a customized zoom area.

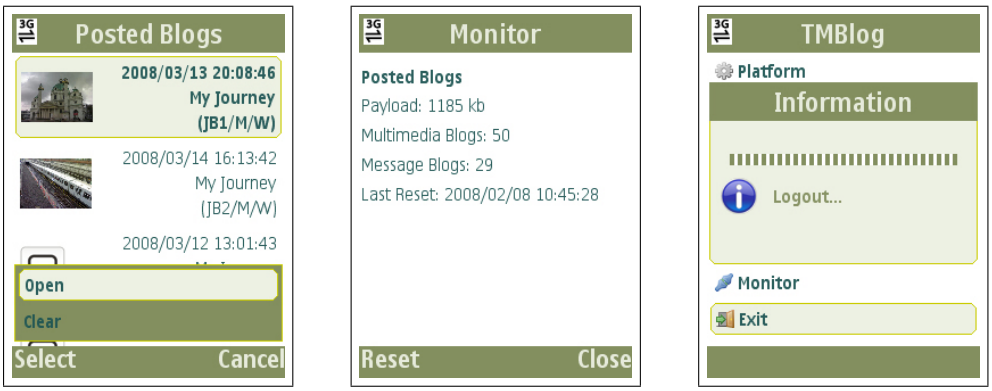


Figure 6.11.: *Screenshots of the Mobile Client - Exploration of the Blogging History. A remaining lightweight Version of the Blog also enables an Exploration of the Blog's Content. This List can be cleared anytime (left). The Traffic Monitor shows the Network Utilization since a specific Date (center). A Logout Operation is performed as far as an User Authentication has been performed on the Server (right).*

Figure 6.12.: *A Nokia N71 handset has been used in the Prototype Testing Phase. An external GNS 5843 Bluetooth GPS Receiver enabled an accurate Determination of the User's geodetic Location.*

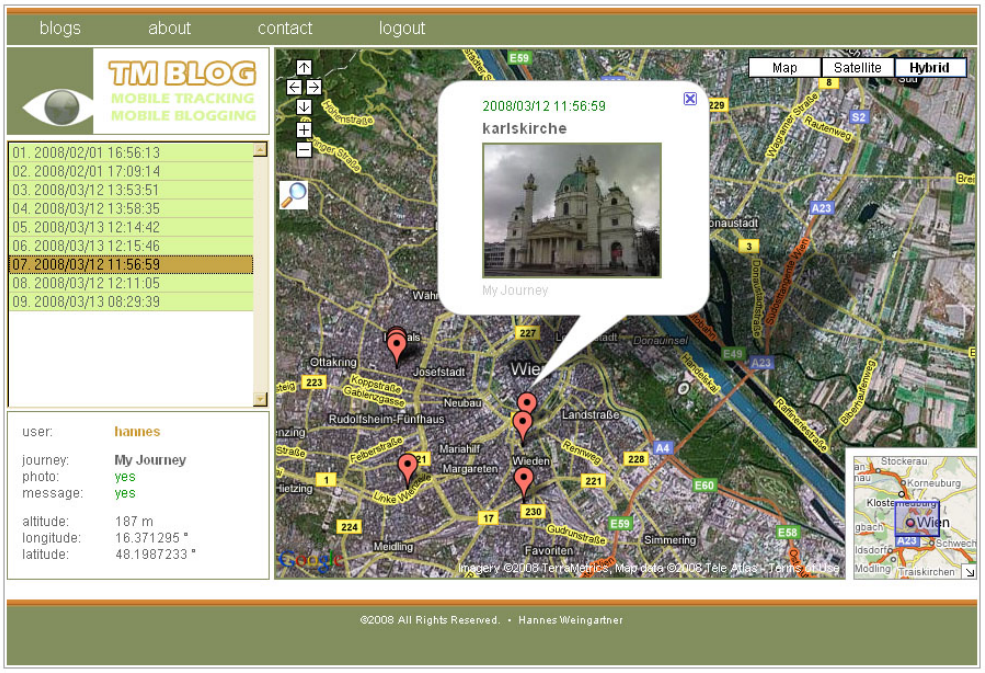


Figure 6.13.: *Screenshot of the Web Client - An opened Photo-related Blog is shown within the Map together with the Time of Creation, an optional Message, and the Journey to which the Blog belongs. The Interface also offers Information about the Waypoint and the Content of the Blog.*

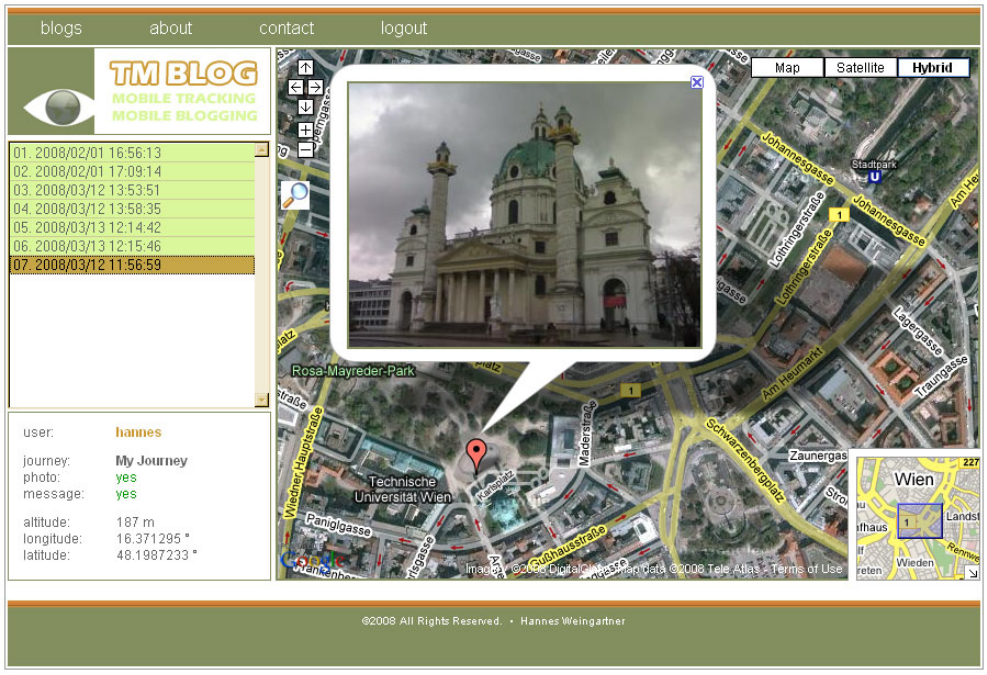


Figure 6.14.: *Screenshot of the Web Client - An opened Photo-related Blog shown at a higher Zoom Level as well as in Full Size. The Photo Dimension for this Instance is 240 x 160 Pixels. Depending on the mobile User's preferences, a maximum Dimension of 480 x 320 Pixels is possible.*

Figure 6.15.: *Screenshot of the Web Client - Registration Form with the Struts Action Errors. Next to the specification of the User's Credentials, a 12-digit Device ID is required.*

7. Related Work and Evaluation

"Thus times do shift, each thing his turn does hold;
New things succeed, as former things grow old."

— Robert Herrick (1648)

Mobile content generation, -processing, and -integration with remote nodes are topics of several academic research projects and evaluation studies today. The aim of related studies and their practical verifications are often based on capability measurements, content adaption, or intelligent interaction strategies within mobile communities. In the following, *three* related academic works and *two* commercial works are briefly presented to offer a classification of the master's thesis and its application field. Furthermore the presentation and the evaluation of related projects shows the differences and the disadvantages compared to the *TMBlog* solution. Related to *TMBlog*, the presented projects are either focused on *Community*- or *Personal Data Management* aspects.

7.1. Academic Approaches

- **Location-based Mobile Blogging**

  The *LocoBlog* project of William Bamford, Paul Coulton, and Reuben Edwards [BCE06] from the *Informatics Group - InfoLab21* at the *University of Lancaster* is similar to the *TMBlog* prototype implementation as part of this thesis. Their implementation however differs in terms of *Privacy*, *Mobile Content Management*, as well as in the underlying *Programming Languages*, and the utilized *Communication Models*. As described in the according paper, this mobile *Python/JavaME* application focuses on more precise location information like the actual *Cell ID*, *GPS Receiver Information*, and on-device *Compass Bearing*. The blog transfer to the server is based on *HTTP Form-encoded Data* which is parsed through an according *PHP* script. Users are tracked by registering the device's 15-digit *IMEI* address, the cartographic web access to an arbitrary user's blogs is however not restricted. The *LocoBlog* application also offers a *3D Space-Time Mapping Plot* of posted blogs. Spatial- and temporal correlated blogs are not available for mobile users. Location awareness offered through area-specific location names is missing too. The absense of mobile web services within this project results in a tightly coupled technology-dependent interaction strategy, which does not scale well.

- **Media Content Metadata and Mobile Picture Sharing**

  Risto Sarvas [Sar04] from the *Helsinki Institute for Information Technology (HIIT)* at the *Helsinki University of Technology (HUT)* presents in his paper a combination of a

Mobile Picture Sharing System and a *Mobile Metadata Creation System*. With a focus on *Personal Media Data Management*, generated content is adapted to be processable with the aim of retrieving and sharing media assets more efficiently. The necessary semantic information origins from a combination of user-defined contextual and social metadata, and descriptive ones which are automatically retrieved by the system. *Location-Unaware* pictures are organized in folders and published directly to a *Web Album*, which is also suitable to be loaded from mobile browsers. In the course of a picture posting process, the application allows to send notifications to persons stored on the phone's address book. In this sense, the application has been designed as a management tool for *Social Activities* within mobile communities. The web interface additionally offers a *Calendar* feature for tracking activities of certain user groups. In contrast to *TMBlog*, this project is strongly related to the mobile browsing paradigm, which typically results in high volume payloads and a decreased responsiveness because page layout and rendering information have to be transfered as well. Since the generated media content is not geotagged, a further location-based processing of user-centric data is not possible.

- **Usage Patterns of FriendZone**

  FriendZone is a suite of *Mobile Location-Based Services (LBS)* for virtual communities with a focus on extended communication patterns and privacy management [BS04]. This academic solution stands under the hood of the *Carnegie Mellon ETC* and the *MIT Media Laboratory*. 47.000 users tested the according services such as *LBS Chat*, *Instant Messaging and Locator (IM&L)*, *Anonymous Instant Messaging (AIM)*, etc. One aim of this test and the according interviews with the subscribed testers was a verification of the acceptance of LBS' through mobile users. Service design concepts, privacy, and interaction patterns on low-level graphics devices have been important guidelines during this evaluation. *FriendZone* is a multi-platform approach involving mobile and static communication nodes. The look-and-feel of integrated services stays the same across domains. According to [BS04] *'users are able to manage buddy lists by adding friends, based on their approval, using phone numbers as identifiers. They can then view their buddies' enhanced virtual presence, send them textual messages which they receive instantly, and view their location'*. The location information might either be obtained in an absolute way by using the network's CellID or as a relative distance to the user, indicated through special ASCII characters. On 3G devices and on *Personal Computers (PCs)* with more capable user interfaces, the buddy list is presented in a *Radar-like Graphical Map* including *Distances* and *Compass Directions*. A similar, but commercial approach is *Qiro*[1]. With *Qiro* the own location can be retrieved within a mobile map immediately. Friends in the near surrounding can be discovered in a similar way through a buddy list with according distances. Furthermore it is a location-based information and navigation system to be able to explore the near surrounding including streets, ATMs, shops, etc. *FriendZone* and *Qiro* are not directly related to mobile blogging, but the LBS-based community- and the multi-domain aspect allow a comparision with the *TMBlog* approach.

[1] https://www.myqiro.de/web/

7.2. Commercial Approaches

- **Lifeblog**

  Lifeblog is a free commercial tool developed by *Nokia*. The software is shipped with N-Series handsets. *Lifeblog* is a mobile diary which might be used to create and to combine personal geotagged messages and media data on the phone[2]. The according blogs might be transfered on a PC for managing and organizing the entries within a calendar or an album with one virtual diary page per day. This concept is similar to *MobShare* (*Media Content Metadata and Mobile Picture Sharing* [7.1]). The access to this user-centric data is however restricted to the content owner exclusively. In this sense it is no community-based tool. This solution also offers a backup function for the whole diary. Composed blogs related to the *TMBlog* project are treated in a similar way on the mobile platform and posted Blogs might also be accessable by the author exclusively. Unlike as with the static part of *Lifeblog*, the web application related to *TMBlog* is used for an exploration of posted blogs only, without any backup- or manipulation function in the prototype software release.

- **TrailExplorer**

  TrailExplorer is an open-source mobile Java approach for recording and viewing trails on smart phones, which is useful for hikers, bikers, and outdoor activities in general. The software is a commercial product developed by Tommi Laukkanen[3]. Trail-related waypoints are automatically recorded in user-defined time intervals, optionally along with manual annotations to mark specific places or events. The according geodetic information is stored within a KML or GPX file. These file formats are used to import tracks in the *Google Earth*[4] application. This PC software might be used to re-explore a trail in a highly interactive cartographic environment. Furthermore the created files might be archived as part of an athletic training program to be able to observe his own performance. *TrailExplorer* is a useful software installable on many smart phones. It offers mobile features like compass bearing, elevation level schemata, distance information, GPS satellite statistics, etc. Compared to the *TMBlog* project, *TrailExplorer* is a personal tracking tool exclusively, without any communication features. Since the recorded files can be easily transfered by using standard communication channels, trail-specific data can be exchanged between different parties in this way.

7.3. Prototype Testing

The *TMBlog* application as a whole has been tested in different development stages, followed by several revisions. The final prototype implementation has been proven as reliable distributed system with a benefit for various scenarios. An exhausting *Sightseeing Tour* through the city of Vienna has finished the prototype testing phase. Content quality, performance, and location accuracy were satisfying and the application handling has been verified as intuitive through independent test persons.

[2] http://www.golem.de/0603/43995.html
[3] http://www.substanceofcode.com/software/mobile-trail-explorer/
[4] http://earth.google.com/intl/de/

- The access to *Location Services* is straightforward and once integrated, subsequent location inquiries are possible in intervals of a few seconds, which enables *Tight Mashups of Waypoints* on the map. The MIDlet's *Trace Management* feature is working fine and the measured distances are realistic. Depending on the chosen measurement unit, the perceived waypoint resolution on the device is however restricted to 100 meters/feet. Determining logical location information through the involved *ISP* has been proven as fast, reliable, and as an useful feature for a *Rough Awareness of Places* during a journey in foreign areas, like districts within large cities or countryside-subdivisions. The application has also been tested successfully on the GPS-capable Nokia N95 handset without deployment or execution problems, like conceived.

- It also came to light that the utilization of SOAP-based *Mobile Web Services* causes no considerable usability limitations even when transfering messages with binary attachments up to 90KB of data. In most of the cases, *Remote Connections* have been quite stable and a high availability of 3G networks furthermore resulted in a high data throughput. Due to this fact, estimated durations for remote invocations like defined as upper boundaries in figure 5.6, have by far not been exceeded. The web service approach also offers benefits in terms of *WSLA Extensions* and easy adaptions of the service interface for future requirements.

- The representation of posted blogs in a map punctuates the contextual significance of a specific location. Photos with a maximum dimension of 480 x 360 pixels seem to be a proper size for loading them within the *Map* and to mediate impressions of distinct places and objects, as they are represented within the blogs.

7.4. Problems

During the prototype implementation- and testing phase some problems arose, which led to an adaption of the software design or to steady usability restriction in the final software revision.

- Since network performance and -responsiveness are varying metrics in the wireless domain, the design finally considered this fact through a loosely coupling with server components. Due to this change the application might be used with nearly all its features without the need of being logged on. *Network Connectivity Problems* during the authentication and posting phases are rare, but occur frequently. In such cases the user is informed, and a subsequent attempt usually succeeds.

- Device-specific metrics like the unavailability of hardware features caused by *Erroneous Vendor Firmware Updates* prevented the MIDlet from successfully accessing the capabilities of the built-in camera on the Nokia N71 handset. Since no firmware downgrades can be performed by the device owner, the mobile host was not suitable any more to fulfill the requirments from the prototype analysis phase. As a consequence the handset had to be exchanged.

- The decision of *Utilizing the RMS Data Store* on the device to enable an easy deployment on all KVM handset without the need of installing additional DB systems, also led to problems. For one thing, accessing and manipulating stored entries is inconvenient and resource-stressing if many entries are affected, and for another thing the native

implementation does not work well. The problem is that removed entries still occupy the storage, which might be a problem for less capable handsets. Only rejecting the whole store frees the hardware resources.

- The *J2ME Polish* mobile application framework is supporting the development process in a great measure, but some outstanding bug fixes led to *Consistency Problems in the GUI*. More exactly, the display mode for the video control is restricted to the raw canvas, which results in an unavailability of form-specific widget layouts. The usage of Symbian's look-and-feel during the photo capture phase is therefore obligatory.

- Since the *GSM cells* in Austria and in many other countries are rarely mapped yet for supporting LBS systems, user-side GPS receivers are essential for public location-based applications. A Bluetooth-capable receiver has to be situated within a radius of approximately five meters related to the handset. Within this area a manual discovery of the device is usually no problem. Based on the environment more attempts might be necessary, until the receiver can be chosen from the offered application-specific device list. The establishment of an initial Bluetooth service connection to the GPS unit might also fail on the first attempt. Depending on the weather conditions, the *TTFF* value of the receiver lies between ten seconds and several minutes. Near to windows active satellite links are also possible within buildings, which however sometimes result in more inaccurate geodetic data with a derivation up to a few hundred meters.

- Even if the mobile device is strongly seen as an agile terminal for the content aggregation in a web context, its on-device content management -and waypoint tracking features resulted actually in a fat client implementation. The drawback of this strategy is that memory issues have to be considered in a greater measure on the mobile platform. The prototype testing phase showed that the batch-based processing of ten photo-related blogs and more causes *Resource Problems*. This is a steady fact for the prototype, whereby the processing capabilities heavily depend on the platform's hardware and the amount of concurrently running processes. The application however does not freeze, the user is just informed that the operation can not be performed. Memory problems might be solved by avoiding caching of blog set by the application, which can be disabled easily through minor code changes. In such cases more efficient mobile database systems would probably compensate a degraded responsiveness due to missing caching facilities.

7.5. Comparison

In the following, the introduced related academic and commercial software approaches are evaluated in terms of their main application fields, actuality, technology, and features. Figure 7.1 gives and overview of these approaches. This evaluation enables a direct comparison between the mentioned solutions. *TMBlog* is a mobile and simple to use location-based data management and asynchron communication tool. Due to these characteristics, the solution might be interesting for tourists. Tourists are typically in a steady movement, they communicate with people at home and they document their activities by writing entries in diaries or by taking pictures of themselves, expressive places, buildings, etc. Software solutions for the tourism sector are typically related to assisting guides, booking systems, weather forecasts, and further services [WK99]. The *TMBlog* solution might be seen as a personal data management tool with roots in both, the classical *Tourism* and the virtual *Community* sector.

Approach	Major Focus	Technology	Application	Project Owner	Latest Release
Locoblog	Location Determination Mobile Content Delivery Map-based Exploration	Python JavaME PHP, JS	Location-based Mobile Blogging	Mobile Radicals	2006
Mobshare	Web Album Mobile Content Delivery Activity Management	Symbian C++ JavaEE Struts	Mobile Picture Sharing	HIIT Finland Futurice	2004
FriendZone	Location-based Chat Instant Messaging Privacy Management	n/a	Location-based Community Services	AxisMobile Carnegie Mellon ETC MIT Media Laboratory	2004
Lifeblog	Personal Multimedia Data Management Mobile Content Delivery	Symbian C++	Personal Multimedia Diary	Nokia	2006
TrailExplorer	Location Determination Mobile Trail Visualization Google Earth	JavaME XML	Open Source Location-based Mobile Tracking	Tommi Laukkanen	2008
TMBlog	Location Determination Mobile Content Delivery Map-based Exploration	JavaME JavaEE, Struts JS, XML	Location-based Mobile Blogging		2008

Figure 7.1.: *Comparison of related Content-Sharing and Location-based Services in the Mobile Domain. The TMBlog Project refers to the Software Solution, which is designed and implemented in the Course of this Master's Thesis. The Table shows related academic and commercial Approaches within this Application Field.*

Next to a high level overview of related solutions, four main *Criteria* have been defined to enable a closer focus on implemented features: *Security*, *Context*, *Communication*, and *Usability*. These criteria are derived from a study on mobile IT solutions for the tourism sector at the *University of Auckland* in 2005. The study is based on location-based services for travellers and is outlined in section *Tourism-related Studies and Deployment* [4.1]. Figure 7.2 shows an evaluation of the mentioned approaches according to these critera.

		Security				Scope of Context												Communication					Usability			
		Authentication																Content Delivery								
		Mobile	Web	Encrypt Channels	Photo	Short Messages	Time-dependent Media	Time	Geodetic Location	Logical Location Information	Calendar	Web-based Map Interface	On-Device Trace Exploration	Trace File Delivery	On-Device Blogging History	Simple Navigation	Visual Blog Statistics	Manual	Semi-Automatic	Service-Orientation	Bidirectional	Community Notification	Extended Mobile User Interface	MVC Strategy	User Preference Settings	Platform Awareness
Approach	Locoblog	✔			✔	✔		✔	✔			✔				✔	✔	✔	✔			✔			✔	
	Mobshare	✔	✔		✔	✔		✔			✔							✔			✔	✔			✔	
	FriendZone	✔	✔			✔			✔									✔			✔	✔			✔	
	Lifeblog				✔	✔	✔	✔	✔	✔	✔							✔	✔						✔	
	TrailExplorer					✔		✔	✔				✔	✔		✔									✔	
	TMBlog	✔	✔	✔	✔	✔		✔	✔	✔		✔	✔		✔			✔	✔	✔			✔	✔	✔	✔

Figure 7.2.: *Comparison of the introduced Approaches in Terms of Security, Context, Communication and Usability.*

The main differences and main advantages of *TMBlog* compared to other approaches is a focus on *Security*, *Usability*, *Portability*, and a loosely coupled integration of data and functionality from multiple domains.

8. Conclusion

"At the end its only counting what we have done and experienced,
and not what we have desired."

— Arthur Schnitzler (1862-1931)

This master's thesis discussed mobile platforms and communication models to efficiently integrate mobile solutions into distributed software infrastructures by considering physical environment parameters. As a practical verification of related technologies, a *Location-Aware Mobile Blogging and Tracking Solution* named *TMBlog* has been designed and implemented. *TMBlog* offers a social tool within the *Tourism Sector* with useful features for managing user-centric mobile geotagged data by considering privacy and usability aspects in a great measure. Additionally, the *TMBlog* service is running on a variety of smart phones since the implementation is based on Java. This solution aims to adapt useful and proved features, as they are implemented in academic and commercial approches as discussed in chapter *Related Work and Evaluation* [7]. Additionally, this mobile solution offers a flexible interaction with web users and the underlying technical infrastructure, while utilizing mobile frameworks and supporting multi-domain services.

The prototype might be extendend through a *Messaging Framework* with bi-directional communication capabilities to receive feedback from a known community over various channels like SMS, MMS, E-Mail or HTTP. Since the application is mainly seen as a tool for a flexible aggregation of geotagged content within a web-based context, a processing of blogs through further services in form of technology-independent *XML-Representations* offers a vast number of additional possibilities. On most of the problems mentioned in the section *Problems* [7.4], the application developer has hardly no influence, because they are based on mobile infrastructure implementations and technology-intrinsic limitations. However a suitable application design and a consideration of platform characteristics and their service environments, enable general rich mobilized software systems towards application scenarios which may benefit from upcoming mobile technologies. With the establishment of *GPS-capable Devices*, the LBS infrastructure and the revenue from such systems will grow, leading to new XML standards for an automated processing of *Geotagged* artifacts which are widely distributed over the world within the wired- and the wireless domain. Location-related descriptions and object-specific metadata information, used for characterizing, organizing, and searching resources will be essential requirements within the *Semantic Web*. Additionally, a wide area *WiMAX* deployment in correlation with operator networks and sophisticated vendor-supported *Service Platforms* have the potential to enable reliable pervasive computing at high data rates. The success of such setups is also strongly related to moderate *Service Fees* in the end-user segment. The utilization of various services in the mobile domain will increase steadily and future setups will include capable mobile devices acting as providers for data and functionality, manifested as high dynamically incorporations in *All-IP* networks.

Appendices

A. Organizations

"The nice thing about standards is that you have so many to choose from. Futhermore, if you don't like any of them, you can just wait for next year's model."

— Andrew S. Tanenbaum

This section describes standardization organizations related to the mobile IT sector. The mobile value chain is represented by operators, handset manufacturers, chip providers, content providers, and application and OS developers. To ensure the highest possible interoperability between these IT groups also in terms of converging the mobile and fixed Internet, standardization bodies are essential. The following briefly presented organizations have a major impact in the evaluation of mobile platforms including security and interoperability issues between mobile and static communication systems by recommending and releasing industry wide technology standards. The primary work is therefore usually the establishment of openess within the corresponding market. For this reason a lot of these organizations' work also concentrates on marketing aspects including press and conference talks.

A.1. Open Mobile Alliance (OMA)

In the year 2002 the *Open Mobile Alliance* (OMA) was founded with the purpose to force more openness within the mobile IT market. The main focus is to bring more insight into the technical aspects of products from corresponding companies. OMA members are therefore advised to provide public lists of products including specified OMA mobile services like interoperability and device capability issues. The so called *OMA Technical Plenary* supports a deployment of rich mobile applications and services by delivering certificated technical specifications to applications and frameworks [OMA07]. Prime objectives of OMA are standardardization efforts in Mobile Web Browsing (e.g. mobile profiles for XHTML, ECMAScript, WML and CSS), Data Synchronization, Multimedia Messaging, Web Services, and *Digital Rights Management* (DRM) [Mob07].

A.2. Open Mobile Terminal Platform (OMTP)

The operator driven *Open Mobile Terminal Platform* (OMTP) Alliance was founded in 2004 and aims at standardizing and at recommending guidelines effecting the usability, device management, and security issues for mobile systems [OMT07]. OMTP for instance publishes operator recommendations for remote service provisioning as well as for device management, SIM and VoIP settings.

A.3. World Wide Web Consortium (W3C)

The goal of the *World Wide Web Consortium* (W3C) is to build and maintain an universal access to the web, a semantic web, and a web of trust. W3C defines its role in a design and a standardization leadership to achieve a decentralized machine interaction and processing including security mechanisms. The consortium is working in the areas *World Wide Web* (HTML), *Data Modelling* (XML family), *Web Services* (SOAP, WSDL), *Security* (XML Signature and XML Encryption), *Semantic Web*, and *Resource Description Framework* (RDF) [W3C07]. Many W3C specifications are the foundation for further interoperability standards, introduced through organizations with a focus on more concrete aspects, like web services (e.g. OASIS, WS-I, and Liberty Alliance) [HKI06].

A.4. OASIS

An important player devoted to the development of standards within the e-business area is the *Organization for the Advancement of Structured Information Standards* (OASIS). Started with a focus on SGML the organization broadened its scope to XML and related standards in 1998. The work is split in so called *Technical Commitees* (TCs), which are initiated by OASIS members including formerly independent standard organizations (e.g. LegalXML, PKI, UDDI). TCs are responsible for industry standards like *SOAP Message Security* (WS-Security) and *Security Assertion Markup Language* (SAML) [HKI06].

A.5. Web Service Interoperability (WS-I)

The *Web Service Interoperability* (WS-I) tries to ensure web service interoperability across platforms, applications, and programming languages. Technical working groups are *Basic Profile* (SOAP 1.1, WDSL 1.1, UDDI 2.0, SOAP Messages with Attachments), *Basic Security Profile*, *Requirements Gathering*, *Sample Applications*, *Test Tools*, and *XML Schema Work Plan*. WS-I offers best practice methods, usage guides, service analysis and optimization, and multi-vendor demonstrations of service interoperability [HKI06].

A.6. Parlay

The parlay consortium consists of Internet service vendors, software developers, leading IT companies, network providers, etc. An important target is to link IT applications with the capabilities of the telecommunication systems through feature rich, simple to use and secure APIs. The resulting specification are base on open standards to support service and solution providers to utilize mobile operator infrastructures. This strategy should enable IT developers to access and explore infrastructures of network- and service providers to build open and advanced B2B and B2C applications [Par08].

A.7. Liberty Alliance

Since 2001 the *Liberty Alliance* plays a crucial role in enabling distributed identity management for sensitive services like implemented in business and social applications. Liberty Alliance therefore covers technology, privacy, and e-business issues. The members consist of government organizations, IT vendors, system integrators, and end-user companies. The alliance addresses the demand for identity federation to share identity information across networks and organizational bounderies in a secure way while keeping a suitable service interoperability. This is a very important topic in every environment to prevent the abuse of personal information by unauthorized system users. Due to the usage of sensitive services like location and contact information of the user, a reliable authority based identity management within a mobile infrastructure is one of the most important aspects when maintaining a pervasive architecture. The Liberty Alliance is currently the *only* open body addressing this issues. The complete specification package of Liberty Alliance consists of the major elements *Liberty Identity Federation Framework* (ID-FF), *Liberty Identity Web Service Framework* (ID-WSF), and *Liberty Identity Service Interface Specification* (ID-SIS). Nokia's web service implementation on smart phones is based on these frameworks by default to ensure a secure communication between distributed services [HKI06].

B. Mobile Platforms

B.1. Platform Security Capabilities

There are several criterias to provide developers access to handset functionality, categorized in following three capability groups [Hea06]:

1. **Basic Capabilities**. In order to use this functionality a standard application signing with *Symbian Signed* is sufficient.

   - `LocationServices`. Accessing local services and transfer data over the serial port, USB, IR, and point-to-point Bluetooth profiles short-link connections. Does not allow IP over routable profiles (e.g. file transfer, data synchronization with PC).

   - `Location`. Accessing data giving the location of the phone (e.g. cell ID).

   - `NetworkServices`. Accessing remote services without restriction on physical location. This capability controls access to services delivered over GSM, CDMA, and all IP transport protocols including IP over Bluetooth (e.g. voice calls, Internet services).

   - `ReadUserData`. Accessing confidential user data for reading (e.g. contacts, short messages).

   - `UserEnvironment`. Accessing live data about the users environment (e.g. audio, picture, video, and biometric recording).

   - `WriteUserData`. Accessing confidential user data for writing (e.g. deleting user data, storing captured media).

2. **Extended Capabilities**. In order to use this functionality a standard application signing with *Symbian Signed* and several detailed declarative statements by the submitter, explaining why a distinct API must be accessed, is needed.

   - `PowerMgmt`. Ability to kill processes and to control power management (e.g. standby, shutdown).

   - `ProtServ`. Allows a server process to register with a protected name.

   - `ReadDeviceData`. Accessing confidential network operator, mobile phone manufacturer, and device settings for reading.

- `SurroundingsDD`. Accessing distinct logical device drivers for getting input information about the surroundings of the phone.
- `SwEvent`. Ability to simulate UI events and to catch them from any program.
- `TrustedUI`. Creating trusted UI sessions and displaying dialogs in a secure UI environment.
- `WriteDeviceData`. Accessing confidential settings to control the behavior of the device (e.g. device lock, system time).

3. **Phone Manufacturer Approved Capabilities**. In order to use this functionality an authorization of a phone manufacturer or channel certifier is needed.
   - `AllFiles`. Read access to the entire file system and write access to private processes' directories.
   - `CommDD`. Direct access to all device drivers used for communication (e.g. Wi-Fi, USB, serial port).
   - `DiskAdmin`. Access to the file system administration operations (e.g. mounting file systems)
   - `Drm`. Access to *Digital Rights Managment* (DRM) protected content.
   - `MultimediaDD`. Access to multimedia device drivers and APIs.
   - `NetworkControl`. Ability to modify and access network protocol controls.
   - `Tcb`. Write access to executables and shared read-only resources.

C. Service-Oriented Computing

C.1. Standard OSGi Services

1. **Framework Services**

   - `Permission Admin`. Service for manipulation bundles by setting permissions.
   - `Conditional Permission Admin`. Extends `Permission Admin` and allows an operator to set permission only when a distinct conditions are true or false. For instance, a mobile operator permitts operations when the mobile device is in a specific region.
   - `Package Admin`. A service that provides information of the current package sharing state and allows a refresh by recalculating its dependencies.
   - `Start Level`. Defines the order of initializing and starting packages by assigning packages to start levels.
   - `URL Handlers`. Service to dynamically contribute new scheme or content handlers to the URL classes.

2. **System Services**

   - `Log Service`. Receiving and dispatching log entries from and to bundles that subscribed to this information.
   - `Event Admin`. A flexible publish and subscribe mechanism for synchronous and asynchronous events.
   - `Device Access`. A plug-and-play service to match a driver to a new device and automatically download a bundle including an implementation of this driver.
   - `User Admin`. A service supporting an user database for authentication and authorization purposes.
   - `IO Connector`. Extends the *Generic Connection Framework* by new and alternative protocol schemes.
   - `Preference Service`. Access a hierachical database of properties, similar to the Java Preference Class.

- `Device Management Tree`. Provides an abstract tree with device specific management information similar to the OMA DM protocol.
- `Deployment Admin`. Deploys multiple artifacts from one file.
- `Application Admin`. A model whereby applications can get registered and activated on demand.
- `Monitoring`. Service for providing performance data of bundles.
- `Foreign Applications`. This service offers non-OSGi applications like MIDlets an API to access OSGi specific functions.

3. **Protocol Services**
   - `HTTP Service`. A servlet runner, used by bundles to provide functions accessable over an HTTP server, which can be smoothly updated with new servlets.
   - `UPnP`. Maps devices within an *Universial Plug-and-Play (UPnP)* network to the Service Registry and OSGi services to an UPnP network as well.
4. **Miscellaneous Services**
   - `Wire Admin`. Alternatively to the rules defined by bundles to find collaborative services, this service connects different services together according to the producer-consumer service that interchange objects over the wire.
   - `XML Parser`. Allows a bundle to locate a parser with desired properties and compartibility to JAXP.
5. **Programming Services**. Services to better address to additional complexity origined from the dynamic OSGi platform behavior.
   - `Service Tracker`. A class that tracks the services for an application.
   - `Declarative Services`. Uses the OSGi subsystem *Component Service Runtime* to read XML declarations from a bundle with service registration and dependencies, to initialize a it only if really needed. This reduces the footprint of a device.

D. MIDlet Core APIs

D.1. Controller

- `void startApp()`
  Native method. Called by the platform to initialize the application. Contains the functionality fo the startup sequence.

- `void pauseApp()`
  Native method. Called by the platform's higher priority processes to interrupt the MIDlet execution. On some devices this function is not working like expected, whereas a workaround would be necessary. In the prototype this fact is not considered.

- `void destroyApp()`
  Native method. Called by the platform if the MIDlet has to be destroyed. Used to perform terminating tasks.

- `Display getDisplay()`
  Returns the application's display.

- `Displayable getScreen(int scrID)`
  Returns a screen indicated by an application-specific ID.

- `void show(Displayable screen)`
  Used to display a specific screen on the device.

- `void show(String title, InfoCollector collector)`
  Used to display the title and the platform capability values of the category specified through the `InfoCollector`.

- `void show(int type, String text, CommandListener controlScreen, Displayable nextScreen)`
  Used to display an alert of a specific `type` with the content `text`. The specified `controlScreen` is listening for the associated alert commands and the `nextScreen` parameter defines the next visible screen after the alert has been closed.

- `void loadResources()`
  Used to load all application-specific resources at startup.

- `void confirmExit()`
  Used to confirm the user whether the exit operation should be performed.

- `void exit()`
  Used to exit the application.

- `boolean storeSettings(Settings settings)`
  Used to store the system settings in the RMS. Returns `true` if the operation was successful.

- `Settings retrieveSettings()`
  Used to retrieve the system settings from the RMS. Returns the `Settings`.

- `Settings getSettings()`
  Used to get the cached system settings. Returns the cached `Settings`.

- `void storeAsDefaultBTD(int context, BluetoothDevice device)`
  Used to store a discovered bluetooth device as the default device in the RMS within the given `context`. The context might be GPS or BTGateway.

- `boolean storeCredentials(Credentials credentials)`
  Used to store the user credentials in the RMS. Returns `true` if the operation was successful.

- `Credentials retrieveCredentials()`
  Used to retrieve the user's credentials from the RMS. Returns the `Credentials`.

- `Credentials getCredentials()`
  Used to get the cached user's credentials. Returns the cached `Credentials`.

- `void authenticateUser(Credentials credentials)`
  Used to authenticate an user with the given credentials at a remote server.

- `void setLoggedOn(boolean logged)`
  Used to indicate whether the user is logged on a remote node.

- `boolean isLoggedOn()`
  Returns `true` if the user is logged on a remote host.

- `void createBlog(int type)`
  Used to create a blog of a specific type.

- `void loadStoredBlogs()`
  Used to load all stored blogs to be browsable through an application screen.

- `void loadPostedBlogs()`
  Used to load all posted blogs to be browsable through an application screen.

- `void postBlog(Payload payload)`
  Used to post a blog with the given `Payload` as its content to a remote node.

- `void setPosted(boolean posted)`
  Used to indicate whether the currently processed blog was posted successfully.

- `void retrieveLocation()`
  Used to initialize the location determination process.

- `void setCurrentGPSLocation(GPSLocation location)`
  Used to set the last received device location.

- `GPSLocation getCurrentGPSLocation()`
  Returns the last received device location.

- `Statistics getStatistics()`
  Returns the current statistics of the application.

- `void loadStatistics()`
  Used to load the statistics to be displayed through an application screen.

- `Monitor getMonitor()`
  Returns the current traffic monitor of the application.

- `void loadMonitor()`
  Used to load the traffic monitor to be displayed through an application screen.

D.2. Blog Manager

- `void init()`
  Used to initialize related subsystems.

- `void showBlogs(int browserType)`
  Used to start a blog browser instance in order to access blogs as list items. The `browserType` parameter indicates the browser type.

- `void cleanUp()`
  Used to perform clean-up tasks after a blog creation process has been terminated.

- `void cleanUpPost()`
  Used to perform clean-up tasks after a blog posting process has been terminated.

- `void cancel()`
  Used to cancel active processes.

- `class BlogStoreTerminator()`
  Inner Class. Extends `TimerTask`. Used to schedule the termination of the blog storing process in case of memory problems.

- `void createBlog(GPSLocation location, int type)`
  Used to start the blog creation process. The `location` parameter declares a blog as a waypoint and `type` parameter indicates the blog type.

- `void createPhoto()`
  Used to start the photo capture process.

- `void createMessage()`
  Used to start the message capture process.

- `void prepareBlog()`
  Used to composite all related blog content (location, photo, message).

- `void storeBlog()`
  Used to start the blog storing process.

- `void serializeBlog()`
  Used to serialize the blog content as byte array.

- `void storeSerializedBlog()`
  Used to store the serialized blog content as record.

- `void postBlogs(Stack stackOfBlogs)`
  Used to start a batch-based blog posting process. The *stackOfBlogs* parameter contains the related blogs.

- `void postBlog(Blog blog)`
  Used to start a single blog posting process. The `blog` parameter refers the related blog.

- `void updatePostedBlogs()`
  Used to perform updating tasks after a blog has been posted successfully.

- `int getBlogSize(Blog blog)`
  Returns the size of a specific `blog` in bytes.

- `int getNextJourneyBlogNumber()`
  Returns the next unique number for a journey blog.

- `int getJourneyBlogCount()`
  Returns the count of currently stored journey blogs.

- `int getWaypointCount()`
  Returns the count of currently stored waypoints.

D.3. Photo Manager

- `void init()`
  Used to perform initial tasks.

- `void initVideoSettings()`
  Used to grab the photo parameters from the configuration file.

- `void mandatePhoto()`
  Used to mandate the photo manager to create a photo. Starts all necessary components for the capturing process.

- `void showVideo()`
  Used to open a new video screen.

- `void startCamera()`
  Used to start the camera resource.

- `void stopCamera()`
  Used to stop the camera resource.

- `void takeSnapshot()`
  Used to capture a photo of the current scene according the defined photo parameters.

- `void discardPlayer()`
  Used to discard the capture process. Releases also related hardware resources.

- `void finished()`
  Used to perform subsequent tasks as soon as the capture process has been finished.

- `void restart()`
  Used to re-initialize the capture process.

- `void cleanUp()`
  Used to perform clean-up tasks to prevent memory problems.

- `void cancel()`
  Used to cancel an active photo capture process.

- `class PhotoFormatter()`
  Inner Class. Used to convert the captured raw data into a suitable form.

- `Photo PhotoFormatter.prepare()`
  Returns a `Photo` object with content-specific attributes.

- `Image PhotoFormatter.getPreview(byte[] rawData)`
  Returns a formatted `Image` instance of the given captured `rawData` for preview purposes. The image is scaled to the device screen dimension.

- `Image PhotoFormatter.getThumbnail(Image image)`
  Returns a formatted `Image` instance of the given `image` for thumbnail preview purposes. The thumbnail is automatically scaled to the prefered device list item dimension.

- `void PhotoFormatter.cleanUp()`
  Used to perform clean-up tasks to prevent memory problems.

D.4. Trace Manager

- `void trackWaypoint(long timestamp, GPSLocation location)`
  Used to add a new waypoint to the trace. The `timestamp` parameter refers to a distinct journey blog. The `location` parameter refers to the location data for the waypoint.

- `Waypoint getPredecessor()`
  Returns the predecessor of the current processing waypoint within the trace.

- `int getWaypointIndex(long timestamp)`
  Returns the trace index of the given waypoint. The parameter `timestamp` refers to the waypoint.

- `Waypoint getWaypoint(long timestamp)`
  Returns the waypoint with the given `timestamp`.

- `boolean removeWaypoint(long timestamp)`
  Used to remove the given waypoint from the trace. The `timestamp` parameter refers to the waypoint. Returns `true` if the operation has been successful.

- `String[] getWaypointInformation(long timestamp)`
  Returns the temporal and spatial information to the given waypoint's predecessor as a formatted string array. The `timestamp` parameter refers to the waypoint.

- `boolean removeTrace()`
  Used to remove the current trace. Returns `true` if the operation has been successful.

- `void resolveTrace()`
  Used to resolve all waypoints within the trace to logical location information within a threaded environment. This method accesses the *Geonames* web service to perform this task.

- `boolean isResolvedTrace()`
  Returns `true` if all waypoints of the trace have been resolved.

- `String getJourneyDistance()`
  Returns the current journey distance as formatted string.

- `String getJourneyDuration()`
  Returns the current journey duration as formatted string.

- `int getJourneyWaypointCount()`
  Returns the count of journey blogs which are related to a waypoint.

D.5. Record Manager

- `RecordStore openRecordStore(String storeName)`
  Used to open the specified record store. The `storeName` parameter refers to the store store. Returns the desired `RecordStore` instance.

- `boolean addRecord(RecordStore store, byte[] record)`
  Use to add a new record to the specified store. The `store` parameter refers to the record store. The `record` parameter refers to the record data. Returns `true` if the operation has been successful.

- `Stack getRecords(RecordStore store, RecordFilter filter, RecordComparator comparator)`
  Used to get a specified set of records from the given store. The `store` parameter refers to the store. The `filter` parameter refers to the customized record filter which is applied on the stored records. The `comparator` parameter refers to an optional record comparator. Returns the desired `Stack` of records.

- `boolean updateRecords(RecordStore store, byte[] record, RecordFilter filter)`
  Used to update a specified set of records in the given store. The `store` parameter refers to the store. The `record` parameter refers to the new record data. The `filter` parameter refers to the customized record filter which is applied on the stored records. Returns `true` if the operation has been succesful.

- `int countRecords(int storeType, int blogType)`
  Used to get the count of record of the specified store type and blog type. The `storeType` parameter refers to the store type. The `blogType` parameter refers to the blog type. Returns the count of records[1].

- `int countRecords(RecordStore store)`
  Used to get the count of record of the specified store. The `store` parameter refers to the record store. Returns the count of records.

- `int removeRecords(RecordStore store, RecordFilter filter)`
  Used to remove records from the given store. The `store` parameter refers to the store. The `filter` parameter refers to the customized record filter which is applied on the stored records. Returns the count of records which have been removed successfully.

- `boolean closeRecordStore(RecordStore store)`
  Used to close the specified record store. The `store` parameter refers to the record store. Returns `true` if the operation has been successful.

- `boolean setCredentials(Credentials credentials)`
  Used to store the user's credentials. The `credentials` parameter refers to the credentials. Returns `true` it the operation has been successful.

[1] The store type may refer to a store containing *local* blogs are a store containing *remote* blogs. The blog type may refer to blogs containing a *Message*- or a *Photo*-related content

- `Credentials getCredentials()`
  Returns the stored user's credentials.

- `boolean removeCredentials()`
  Used to remove the user's credentials from the system. Returns `true` if the operation has been successful.

- `boolean setSettings(Settings settings)`
  Used to store the system settings. The `settings` parameter refers to the system settings. Returns `true` it the operation has been successful.

- `Settings getSettings()`
  Returns the stored system settings.

- `boolean setBlog(int storeType, int blogType, Stack recordSet)`
  Used to store a set of blogs. The `storeType` parameter refers to the store type. The `blogType` parameter refers to the blog type. The `recordStack` parameter refers to a set of serialized blogs. Returns `true` if the operation has been successful.

- `Stack getBlog(int storeType, int blogType, int quality, RMSBlogSelector filter)`
  Used to get a specified set of blogs. The `storeType` parameter refers to the store type. The `blogType` parameter refers to the blog type. The `quality` parameter refers to the desired blog quality[2]. The `filter` parameter is an instance of a blog-specific `RecordFilter` class which is used to define query conditions. Returns a set of serialized blogs.

- `int removeBlog(int storeType, int blogType, RMSBlogSelector filter)`
  Used to remove a specified set of blogs. The `storeType` parameter refers to the store type. The `blogType` parameter refers to the blog type. The `filter` parameter is an instance of a blog-specific `RecordFilter` class which is used to define query conditions. Returns the count of serialized blogs which have been removed successfully.

- `boolean setWaypoint(Waypoint waypoint, RMSWaypointSelector filter)`
  Used to store or update a specific waypoint. The `waypoint` parameter refers to the waypoint. The `filter` parameter is an instance of a waypoint-specific `RecordFilter` class which is used to define query conditions. If the filter parameter equals `null`, a new waypoint is stored, otherwise the related waypoint will be updated. Returns `true` if the operation has been successful.

- `Vector getWaypoint(RMSWaypointSelector filter)`
  Used to get a specified set of waypoints. The `filter` parameter is an instance of a waypoint-specific `RecordFilter` class which is used to define query conditions. Returns a set of waypoints.

- `int removeWaypoint(RMSWaypointSelector filter)`
  Used to remove a specified waypoint. The `filter` parameter is an instance of a waypoint-specific `RecordFilter` class which is used to define query conditions. Returns the count of waypoints which have been removed successfully.

[2]The parameter may refer to a *thumbnail* or a *full* version of photo-related blogs.

D.6. Blog Browser

- `void init()`
  Used to perform initial tasks.

- `void browseBlogs()`
  Used to initiate the browsing capabilities and to display the visual blog list.

- `void createItem(Blog blog, Integer listIndex)`
  Used to create a new browser list item. The `blog` parameter refers to the represented blog. The `listIndex` refers to the position within the list.

- `void refreshScreen()`
  Used to refresh the blog list and commands according to the current system state.

- `void refreshItemInfo()`
  Used to refresh the blog item info according to the current system settings.

- `int getBlogIndex()`
  Returns the currently selected index in the blog list.

- `void loadBlogs()`
  Used to load available blog sets into the browser.

- `void updateBlogs()`
  Used to update a loaded blog set.

- `Stack getBlogSet(int blogType, RMSBlogSelector selector)`
  Used to retrieve a set of blogs from a specific blog type. The `blogType` parameter refers to the desired blog type. The `selector` parameter is an instance of a blog-specific `RecordFilter` class which is used to define query conditions. Returns a set of blogs.

- `void openSelectedBlog()`
  Used to open the currently selected blog to view its details.

- `void deleteSelectedBlog()`
  Used to delete the currently selected blog from the system.

- `void deleteBlog(Blog blog)`
  Used to delete a specific blog from the system. The `blog` parameter refers to the blog.

- `void deleteJourney()`
  Used to delete the current journey from the the system.

- `void postSelectedBlog()`
  Used to transfer the content of the currently selected blog to a remote network node.

- `void postJourney()`
  Used to transfer the content of the current journey-related blogs to a remote network node.

- `void resolveJourney()`
  Used to resolve the journey-related waypoints to logical location information.

- `void clear()`
  Used to remove all blogs from the system.

D.7. Bluetooth Manager

- `void init()`
  Used to perform initial tasks.

- `void searchEnvironment()`
  Used to initialize a bluetooth discovery process.

- `void inquiryComplete(int discType`
  Native Method. Called by the system if an inqiry is complete. The `discType` parameter refers to the type of request that was completed.

- `void scheduleTimeoutNotifier()`
  Used to schedule a timer for automatically aborting an active discovery process after a specific amount of time has been elapsed.

- `void inqiryTimeoutNotifier()`
  Inner class. Extends *TimerTask*. Used to abort an active discovery process.

- `void searchDevices()`
  Used to start a bluetooth device discovery process.

- `void deviceDiscovered(RemoteDevice remoteDevice, DeviceClass deviceClass)`
  Native method. Called by the system if a device is discovered. The `remoteDevice` parameter refers to the discovered device. The `deviceClass` parameter refers to the discovered device class.

- `boolean isDeviceSearchComplete()`
  Returns `true` if the device discovery process in complete.

- `void setSelectedDevice(BluetoothDevice device)`
  Used to set the device selected by the user.

- `Vector getDevices()`
  Returns a set of discovered devices.

- `void cancelDeviceSearch()`
  Used to cancel an active device discovery process.

- `void searchServices(RemoteDevice remoteDevice)`
  Used to start a bluetooth service discovery process. The `remoteDevice` parameter refers

to the device which offers services.

- `void serviceDiscovered(int transID, ServiceRecord[] servRecord)`
  Native method. Called by the system if a service is discovered. The `transID` parameter refers to the transaction ID of the service search that is posting the result. The `servRecord` parameter refers to a list of services found during the search request.

- `void serviceSearchCompleted(int transID, int respCode)`
  Native method. Called by the system if a service discovery process is complete. The `transID` parameter refers to the transaction ID of the service search that is posting the result. The `respCode` parameter refers to the response code that indicates the status of the transaction.

- `Vector getServices()`
  Returns a set of discovered services.

- `void cancelServiceSearch()`
  Used to cancel an active service discovery process.

D.8. Location Manager

- `void retrieveLocation()`
  Used to start the according location service access components.

- `boolean isValidDevice(BluetoothDevice device)`
  Used to determine if the specified device is a valid GPS receiver. Returns `true` if the `device` parameter refers to a valid device.

- `void finished()`
  Used to perform final tasks.

- `void cleanUp()`
  Used to perform clean-up tasks to prevent memory problems.

- `void retrieveLocationFromBTD()`
  Used to send a location query to a previous discovered and selected GPS-enabled *Bluetooth Device* (BTD).

- `void retrieveLocationFromGLP()`
  Used to send a location query to a *Generic Location Provider* (GLP). The discovery and the access to this providers is managed through the underlying system.

- `void onLocationInquiryCompleteBTD(boolean available)`
  Used as a callback function after a location inquiry to the BTD has been completed. The location provider's functionality runs within a threaded environment. The `available` parameter indicates if valid location data is available.

- `void onLocationInquiryCompleteGLP(boolean available)`

Used as a callback function after a location inquiry to the GLP has been completed. The location provider's functionality runs within a threaded environment. The `available` parameter indicates if valid location data is available.

D.9. Gateway WSC

- `GatewayServiceConnection getServiceConnection()`
  Returns an instance of the `GatewayServiceConnection` class in order to deliver endpoint-specific connection attributes.

- `void onAuthenticationRequestComplete(String info)`
  Used as a callback function to indicate a completed user-related request. The optional `info` parameter refers to response-specific information.

- `void onAuthenticationRequestError(String error)`
  Used as a callback function to indicate an error for an user-related request. The optional `error` parameter refers to occured error.

- `void onPostRequestComplete(String info)`
  Used as a callback function to indicate a completed blog-related request. The optional `info` parameter refers to response-specific information.

- `void onPostRequestError(String error)`
  Used as a callback function to indicate an error for a blog-related request. The optional `error` parameter refers to occured error.

- `void logon()`
  Used for a server logon, in order to establish a session by verifying the user's credentials and the according mobile device.

- `void logout()`
  Used for a server logout, in order the close a session.

- `void post(Payload payload)`
  Used to transfer the content of a blog to the server. The `payload` parameter refers to the content of the blog.

D.10. Geonames WSC

- `HttpConnection getServiceConnection(String url)`
  Returns the an instance of the `HttpConnection` class in order to be able to send service requests. The `url` parameter refers to the string representation of the service endpoint.

- `void verifyConnection(HttpConnection connection)`
  Used to verify the service reachability. The `connection` parameter refers to the service connection.

- `Toponym getToponym()`
  Returns an instance of the `Toponym` class in order to deliver waypoint-specific location identifiers.

- `void cleanUp()`
  Used to perform clean-up tasks to prevent memory problems.

- `void findCountryName(Waypoint waypoint)`
  Used to determine the country name related to the specified waypoint. The `waypoint` parameter refers to the waypoint.

- `void findNearbyPlaceName(Waypoint waypoint)`
  Used to determine the nearby place name related to the specified waypoint. The `waypoint` parameter refers to the waypoint.

- `void findSRTMElevation(Waypoint waypoint)`
  Used to determine the SRTM-based elevation value related to the specified waypoint. The `waypoint` parameter refers to the waypoint.

- `void parseResponseCountry(InputStream is)`
  Used to parse the server's XML-based response in order the determine the country name. The `is` parameter refers to the response data stream.

- `void parseResponseNearbyPlace(InputStream is)`
  Used to parse the server's XML-based response in order the determine the nearby place name. The `is` parameter refers to the response data stream.

- `void parseResponseElevation(InputStream is)`
  Used to parse the server's XML-based response in order the determine the elevation value. The `is` parameter refers to the response data stream.

List of Figures

List of Tables

Listings

References

[3GA05] *The Java Wireless Application Environment*. http://www.3gamericas.org/pdfs/java_mar2005.pdf, 2005.

[A-S04] A-SIT - *Sicherheitsanalyse - Blackberry mobile data service*. http://www.cio.gv.at/securenetworks/20040112_StudieBlackberry.pdf, 2004.

[BCE06] William Bamford, Paul Coulton, and Reuben Edwards. Location-based mobile blogging. *Information and Communication Technologies, 2006. ICTTA '06. 2nd*, 1:111–116, 2006.

[Bec08] Rolf Becking. Pragmatismus statt Shangri-La. *Java Magazin*, (1.08):50–55, 2008.

[Bia08] Marek Bialoglowy. *Bluetooth Security Review - Part 1*. http://www.securityfocus.com/infocus/1830, 2008.

[Bla06] ThinPrint - *Blackberry Security*. http://www.thinprint.de/uploads/File/TP/White%20Papers/blackberry_security_de.pdf, 2006.

[BM06] Ulrich Breymann and Heiko Mosemann. *JavaME - Anwendungsentwicklung für Handys, PDA und Co.* Carl Hanser Verlag, München, Wien, 2006.

[BS04] Asaf Burak and Taly Sharon. Usage Patterns of FriendZone - Mobile Location-Based Community Services. *ACM International Conference Proceeding Series*, 83:93 – 100, 2004.

[BTQ07] *Ubiquitous Mobility - The Enduring Journey*. http://www.btquarterly.com, 2007.

[Car] Bruce Carney. *Evolving to Symbian OS v9*. http://developer.symbian.com/main/downloads/papers/evolving_toV9/evolving_toV9.pdf.

[CC07] Sae Sol Choi and Mun-Kee Choi. Consumer's Privacy Concerns and Willingness to Provide Personal Information in Location-Based Services. *Advanced Communication Technologies, The 9th International Conference*, 3:2196–2199, 2007.

[Cha06] Suresh Chande. *Mobile Web Services*; University of Helsinki. http://www.cs.helsinki.fi/u/chande/MobileWebServices.pdf, 2006.

[CJ02] David A. Chappell and Tyler Jewell. *Java Web Services*. O'Reilly Media, Sebastopol, CA, 2002.

[DD06] Christoph Dorn and Schaharam Dustdar. *Sharing hierarchical context for mobile web services*. http://www.forrester.com/Research/Document/0,7211,43340,00.html, 2006.

[dJ04a] Martin de Jode. *Getting Started with MIDP Programming on Symbian OS*. http://developer.symbian.com/main/downloads/papers/Midpgetstart/GetStartMIDPv0_3.pdf, 2004.

[dJ04b] Martin de Jode. *Programming the MIDP Lifecycle on Symbian OS*. http://developer.symbian.com/main/downloads/papers/midplifecycle/midplifecycle.pdf, 2004.

[dJ04c] Martin de Jode. *Symbian on Java*. http://media.wiley.com/assets/262/10/SymbianOnJava2_05.pdf, 2004.

[dJT04] Martin de Jode and Colin Turfus. *Symbian OS System Definition*. http://developer.symbian.com/main/downloads/papers/SymbOS_def/symbian_os_sysdef.pdf, 2004.

[FTW07] Thilo Frotscher, Marc Teufel, and Depeng Wang. *Java Web Services mit Apache Axis2*. entwickler.press, 2007.

[GAR00] *GARMIN - GPS Guide for Beginners*. http://www8.garmin.com/aboutGPS/manual.html, 2000.

[geo08] *Geonames Web Service*. http://www.geonames.org/, 2008.

[Gla08] Kay Glahn. Der Android auf dem Handy. *Java Magazin*, (1.08):8, 2008.

[Gut07] Frank Gutmann. *Positionsbestimmung in GSM- und UMTS Netzwerken*. http://www.ks.uni-freiburg.de/download/papers/lbsSS07/PositionGSMandUTMS, 2007.

[Hea06] Craig Heath. *Symbian OS Platform Security: Software Development Using the Symbian OS Security Architecture*. John Wiley & Sons, 2006.

[HKI06] Frederick Hirsch, John Kemp, and Jani Ilkka. *Mobile Web Services - Architecture and Implementation*. John Wiley & Sons, 2006.

[Hol03a] Heiko Holtkamp. *Einführung in Bluetooth*; Universität Bielefeld. http://www.rvs.uni-bielefeld.de/~heiko/bluetooth/bluetooth.pdf, 2003.

[Hol03b] Clemens Holzmann. *Bluetooth in a Nutshell - Context Framework for Mobile User Applications*; Institut für Praktische Informatik - Universität Linz. http://www.pervasive.jku.at/Research/Publications/_Documents/Bluetooth-holzmann2003.pdf, 2003.

[Häß01] Achim Häßler. *Entwicklung einer GPS-Bibliothek in Java*.

http://elib.uni-stuttgart.de/opus/volltexte/2001/813/index.html, 2001.

[ibm06] *Designing mobile Web services*. http://www.ibm.com/developerworks/wireless/library/wi-websvc/, 2006.

[Jav04] *Java Verified Program - The Java Verified Program and MIDP 2.0 Security*. http://javaverified.com/docs/Technical-Topics-01-2.pdf, 2004.

[Jav07] *Java Verified Program*. http://javaverified.com, 2007.

[JCP07] *Java Community Process - Community Development of Java Technology Specifications*. http://jcp.org/en/introduction/faq, 2007.

[JDvT04] Ivar Jørstad, Schahram Dustdar, and Do van Thanh. Evolution of Mobile Services: An Analysis of Current Architectures with Prospect to Future. *Ubiquitous Mobile Information and Collaborative Systems*, (3272):131–136, 2004.

[KBS05] Dirk Krafzig, Karl Banke, and Dirk Slama. *Enterprise SOA - Service Oriented Architecture, Best Practices*. Pearson Education, Inc., New Jersey, 2005.

[KO08] Axel Kossel and Rudi Opitz. Internet handlich - Mobil ins Netz mit dem Handy. *C'T Magazin für Computertechnik*, (3.08):92, 2008.

[Leu05] Clara Leung. *The Perceived Value of Location-Based Services in New Zealand Tourism*; university of auckland. http://www.tourismresearch.govt.nz/NR/rdonlyres/AFD13060-2A1E-4372-84C6-1142FD5B7916/22012/ClaraLeungLBSTourismIndustryReport.pdf, 2005.

[Mob06] *Mobile Open Source*. http://www.funambol.com/blog/capo/2006/11/my-mobile-20-manifesto.html, 2006.

[Mob07] MobEduNet - *International Project for Mobile Systems Programming Education*. http://www.mobedu.net/materialbank.php, 2007.

[MS05] Dillip Mohapatra and Suma S.B. *Survey Of Location Based Wireless Services*. *JCPWC*, pages 358–362, 2005.

[New08] Alan Newman. *Java Bluetooth Object Exchange*. http://developer.symbian.com/main/downloads/papers/SymbianBluetoothOBEXArticlev1.7.pdf, 2008.

[NME07] *National Marine Electronics Association (NMEA) - 0183 Specification*. http://www.nmea.org/pub/0183/index.html, 2007.

[Nok04a] *Forum Nokia - Efficient MIDP Programming*. http://sw.nokia.com/id/d307878f-bbd6-415a-af25-bf7fb3efc9d3/Efficient_MIDPProgramming_v1_1_en.pdf, 2004.

[Nok04b] *Forum Nokia - Designing MIDP Applications For Optimization*. http://sw.nokia.com/id/ff51fcc6-edc0-4763-9874-89527700c7ff/Designing_MIDP_Applications_For_Optimization_v1_0_en.pdf, 2004.

[Nok05] Forum Nokia - *Getting Started with Security*. http://sw.nokia.com/id/2fb09348-acd0-45c1-971f-ccdb626f4218/Getting_Started_With_Security_v1_0_en.pdf, 2005.

[Nok06] *Nokia Forum- Enterprise: Developing End-to-End Systems*. http://www.seap.forum.nokia.com/info/sw.nokia.com/id/7fa55fad-1ebc-43ce-ad42-61b96abec010/Enterprise_Developing_End-to-End_Systems_v2_0_en.pdf.html, 2006.

[Nok07a] *Nokia Forum - Freeware Opportunities for S60 and Series 80 Developers*. http://sw.nokia.com/id/5f6e9bc6-239a-4c8d-81d4-9256c5de1f9c/Freeware_opp_S60_1_1_en.pdf, 2007.

[Nok07b] *Forum Nokia - Remote Device Access*. http://www.forum.nokia.com/main/technical_services/testing/rda_introduction.html, 2007.

[Nok07c] *Nokia for Universities*. http://www.forum.nokia.com/main/forum_nokia_for_universities/index.html, 2007.

[OMA07] *Open Mobile Alliance*. http://www.openmobilealliance.org, 2007.

[OMT07] *Open Mobile Terminal Platform Alliance*. http://www.omtp.org/, 2007.

[OSG07a] *Open Service Gateway Initiative Alliance*. http://www.osgi.org/, 2007.

[OSG07b] *Open Service Gateway Initiative Alliance - About the OSGi Platform*. http://www.osgi.org/documents/collateral/OSGiTechnicalWhitePaper.pdf, 2007.

[Par08] *The Parlay Group*. http://www.parlay.org/en/index.asp, 2008.

[PD04] John Pagonis and Jonathan Dixon. *Location Awareness and Location Based Services - Positioning and Terminology*. http://developer.symbian.com/main/downloads/papers/messaging/LocalAwareness_LBS_01.pdf, 2004.

[Per06] Barbara Pernici. *Mobile Information Systems - Infrastructure and Design for Adaptivity and Flexibility*. Springer Verlag, Berlin, Heidelberg, 2006.

[Qua03] *BREW and J2ME - A complete Wireless Solution for Operators Commited to Java*. http://www.brewpresskit.com/brew/about/brewwhitepaper.pdf, 2003.

[Res02] *REST Web Services*. http://www.oio.de/public/xml/rest-webservices.htm, 2002.

[Res08] *Building Web Services the REST Way*. http://www.xfront.com/REST-Web-Services.html, 2008.

[Rot02] Jörg Roth. *Mobile Computing - Grundlagen, Technik, Konzepte*. Dpunkt Verlag, Heidelberg, 2002.

[Sam08] Bruce Sams. *Single-Sign-On Systeme*. *Java Magazin*, (1.08):15–20, 2008.

[Sar04] Risto Sarvas. *Media Content Metadata and Mobile Picture Sharing*. http://www.seco.tkk.fi/events/2004/2004-09-02-web-intelligence/papers/sarvas_WebIntelligence2versio.pdf, 2004.

[Sch04] Klaus-Dieter Schmatz. *Java 2 Micro Edition - Entwicklung mobiler Anwendungen mit CLDC and MIDP.* Dpunkt Verlag, Heidelberg, 2004.

[Sil07] Chris Silva. *Forrester Research - Mobile Evolution: Moving Toward An All-Wireless Enterprise*. http://www.forrester.com/Research/Document/0,7211,43340,00.html, 2007.

[Sur07] *GPS TTFF and Startup Modes*. http://www.survey-lab.com/, 2007.

[Sym] *What Java Developers need to know about MIDP on Symbian OS*. http://developer.symbian.com/main/downloads/papers/midpjava/WhatJavaDevelopersNeedToKnow_1.0.pdf.

[Sym03] *Why is a different Operating System needed?* http://www.symbian.com/files/rx/file6383.pdf, 2003.

[Sym05] *Symbian OS: Overview To Networking v.1.0*. http://www.forum.nokia.com/info/sw.nokia.com/id/c4536832-3dd0-45af-94be-1c4289cc3003/Symbian_OS_Overview_To_Networking_v1_0_en.pdf.html, 2005.

[Sym07a] *Symbian OS*. http://developer.symbian.com, 2007.

[Sym07b] *Creating the Mass Market for Symbian OS*. http://www.symbian.com/files/rx/file6384.pdf, 2007.

[Sym07c] *Symbian Developer Network - P.I.P.S.* http://developer.symbian.com/wiki/display/oe/P.I.P.S.+Home, 2007.

[Tay05] Mayank Tayal. *Location Services in the GSM and UMTS Networks*. http://ieeexplore.ieee.org/xpl/freeabs_all.jsp?arnumber=1431369, 2005.

[TM06] Kee-Leong Tan and S.M.F.D. Syed Mustapha. *Measuring Availability of Mobile Web Services*. http://iec.cugb.edu.cn/WorldComp2006/SWW4876.pdf, 2006.

[TvS02] Andrew S. Tanenbaum and Maarten van Steen. *Distributed Systems - Principles and Paradigms*. Prentice Hall, Inc., New Jersey, 2002.

[Umt07] *GPRS-Einführung*. http://umtslink.at, 2007.

[Vir05] Robert Virkus. *J2ME Polish: Open Source Wireless Java Tool Suite.* APress Verlag, 2005.

[W3C07] *World Wide Web Consortium (W3C).* http://www.w3c.org, 2007.

[WAP02] *WAP Forum - WAP 2.0 Technical White Paper.* http://www.wapforum.org/what/WAPWhite_Paper1.pdf, 2002.

[Weß06] Matthias Weßendorf. *Web Services & mobile Clients - SOAP, WSDL, UDDI, J2ME, MIDlets, WAP & JSF.* W3L Verlag, Bochum, 2006.

[WK99] Hannes Werthner and Stefan Klein. *Information Technology and Tourism - A Challenging Relationship.* Springer Verlag, Wien, New York, 1999.