

BACHELORARBEIT

Mobiler, persönlicher Assistent

Patrick Hornecker

betreut durch

Klaus Rechert

an der

Technischen Fakultät
der Albert-Ludwigs-Universität Freiburg

Inhaltsverzeichnis

1	Einleitung	2
2	Grundlagen	3
2.1	Aktueller Stand	3
2.2	Ziele	4
2.3	Anwendungsmöglichkeiten	4
2.4	Anforderungen	6
2.5	Verfahren	6
3	Technische Grundlagen	8
3.1	Betriebssysteme für mobile Geräte	8
3.1.1	Windows Mobile	8
3.1.2	Android	9
3.1.3	WebOS	9
3.1.4	iPhone OS	9
3.1.5	Symbian OS	10
3.1.6	Zielplattform	10
3.2	Softwaregrundlagen	10
3.2.1	CeGCC	10
3.2.2	Enlightenment	11
4	Friend Finder	13
4.1	Verwendete Verfahren und Bibliotheken	13
4.1.1	Grafisches Benutzeroberfläche	14
4.1.2	Versenden der Nachrichten	14
4.1.3	Versenden der eigenen Position	15
4.1.4	Empfangen der eigenen Position	16
4.1.5	Erzeugen eines 2D-Barcodes	16
4.2	Analyse	16
4.2.1	Allgemeiner Datenverkehr	17
4.2.2	Versenden und Empfangen von Nachrichten	18
4.2.3	Versenden und Empfangen von Positionen	19
4.2.4	Fazit der Auswertung	19
5	Fazit	20

Erklärung

Hiermit erkläre ich, dass ich diese Abschlussarbeit selbständig verfasst habe, keine anderen als die angegebenen Quellen/Hilfsmittel verwendet habe und alle Stellen, die wörtlich oder sinngemäß aus veröffentlichten Schriften entnommen wurden, als solche kenntlich gemacht habe. Darüber hinaus erkläre ich, dass diese Abschlussarbeit nicht, auch nicht auszugsweise, bereits für eine andere Prüfung angefertigt wurde.

Ort, Datum

Unterschrift

1 Einleitung

Durch den fortschreitenden Stand der modernen Technik ist es möglich immer leistungsstärkere, mobile Geräte zu bauen. So geht die Funktionalität moderner Handys weit über das Telefonieren und Schreiben von SMS hinaus. Aktuelle Modelle, dieser sogenannten Smart Phones, ist es zum Beispiel möglich sich mit einem *WLAN* zu verbinden, die eigene Position mittels *GPS* zu ermitteln oder per *UMTS* Daten zu übertragen.

Aus dieser Fülle an Funktionen und den verschiedenen angebotenen Smart Phones ergibt sich somit eine immense Menge an möglichen Anwendungsgebieten.

Auch positionsabhängige Dienste verbreiten sich, dank *GPS*-Funktionen der Smart Phones, immer weiter. So ist es mit bestimmten Programmen zum Beispiel möglich eine zurückgelegte Strecke zu speichern oder den nächsten Supermarkt in der Nähe anzuzeigen. Dienste dieser Art sind bei Benutzern sehr beliebt und werden auch rege genutzt. Was allerdings von vielen Anwendern übersehen wird, ist die Tatsache, dass es möglich ist durch übermittelte Positionsdaten ein Bewegungsprofil zu erstellen, oder aber dritte Personen könnten die Daten abfangen und anderweitig missbrauchen.

Somit ist es besonders im Bereich von Positionsdaten ein wichtiger Aspekt, Daten zu verschlüsseln. Hierzu müssen Algorithmen genutzt werden, welche das Verschlüsseln von Daten ohne zuviel Berechnungsaufwand durchführen, um den Akku des Gerätes zu schonen. Eine weitere Frage, die sich in diesem Kontext stellt, ist wie man auf sichere Art und Weise die Schlüssel, welche zum verschlüsseln der Daten genutzt werden, sicher und einfach verteilen kann. Hierbei sollte sichergestellt sein dass es für den Benutzer ohne großen Aufwand möglich ist den Schlüssel weiterzugeben und dritte Personen diese nicht abfangen können

Eine solche Software, welche Positionsdaten in einem sicheren Kontext versendet, wurde im Rahmen dieser Bachelor-Arbeit konzipiert und implementiert. Hierbei ging es vor allem um eine sichere sowie dezentrale Datenübertragung. Ein weiterer Teil bildet das Kompilieren eines Programmes für Windows Mobile unter Linux.

2 Grundlagen

In diesem ersten Teil dieser Bachelor-Arbeit wird der momentane Stand von *location awareness* Software auf mobilen Geräten aufgezeigt. Es wird auch analysiert, was für Anforderungen an ein Programm dieser Art gestellt werden um einen sicheren Datenverkehr zu garantieren.

2.1 Aktueller Stand

Da so gut wie alle aktuellen Smart Phones mit einem *GPS* ausgestattet sind existieren für die verschiedenen Betriebssysteme schon eine Reihe von Anwendungen die Funktionalitäten für *location awareness* bieten. So existieren Anwendungen um sich Routen erstellen zu lassen, die eigene Position zu bestimmen oder um *Geocaching* zu betreiben. Es existieren auch eine Menge an Anwendungen die die eigene Position für Freunde sichtbar macht.

So bietet *Google* den Dienst *Google Latitude* [Latitude] an. Bei diesem Programm ist es möglich die Position von Freunden, die diesen auch Dienst nutzen, auf einer Karte anzeigen zu lassen. Es besteht hierbei die Möglichkeit die eigene Position per *GPS* oder mit Hilfe von Daten der *GSM-Funkzelle* zu bestimmen. Es existiert auch eine Paper mit dem Titel *Spontaneous Privacy-Aware Location Sharing* [Rechert und Welke, 2009]. Hierfür wurde ein Dienst für mobile Geräte implementiert, welcher Daten verschlüsselt an mehrere Nutzer sendet. Für diesen Dienst wurde auch mit ein möglichst einfaches und verlässliches Protokoll entworfen, mit dem Ziel die Berechnungszeiten niedrig zu halten. Des weiteren hat man in dem Paper *FriendSensing: Recommending Friends Using Mobile Phones* [Quercia und Capra, 2009] Möglichkeiten erörtert um die Position anderer Benutzer mit Hilfe von *Bluetooth* zu bestimmen. Hierbei registrierte eine Software wie oft welcher Nutzer mit einem anderen in Kontakt stand und wie oft sie in *Bluetooth* Reichweite waren. Diese Aufzählung, über Software die sich mit der Thematik von *location awareness* auseinandersetzt könnte an dieser Stelle noch weiter fortgeführt werden da es hierfür Unmengen an Programmen gibt.

Allerdings nutzen diese Programme, die oft auf *Google Maps* basieren, meistens das Prinzip eines zentralen Knotenpunktes, an welchen die Positionsdaten gesendet werden und dieser diese dann weiterleitet. Somit existiert immer eine zentrale Kontrollinstanz, welche Einsicht in die Daten der Nutzer hat, während die Nutzer selbst immer nur Zugang zu den für sie bestimmten Daten besitzen. Somit können die Nutzer auch nicht die Nutzung ihrer Daten durch den Anbieter einsehen. Die Folgen hiervon könnten sein, dass zum Beispiel der Anbieter gezielt Werbung für die Position der Nutzer einspielt, da er ihren Aufenthaltsort immer kennt.

Bestehende Software dieser Art verschlüsselt auch in den seltensten Fällen die versendeten Daten. Da es sich bei Positionsdaten um sehr sensible Daten handelt stellt dies ein großes Manko dar. So könnten Dritte den Datenverkehr abhören und auch Positionsdaten von Nutzern erhalten,

die diese nur einer bestimmten Gruppe zur Verfügung stellen wollten. Sind diese Positionsdaten erst einmal gesammelt können ohne weiteres Bewegungsprofile erstellt und missbraucht werden. Der Benutzer begibt sich also mit der Nutzung von solchen Programmen in die Gefahr das regelmäßige Aufenthaltsorte erkannt werden und der ständig aufspürbar ist. Somit stellen solche Dienste, die sensible Daten dieser Art ohne Verschlüsselung versenden, eine starke Einschränkung für die Privatsphäre dar.

2.2 Ziele

Die obigen zwei Punkte wurden bisher in den meisten noch nicht berücksichtigt. Somit besteht seitens der Nutzer mit Sicherheit eine Nachfrage nach einer Software, welche ihre Persönlichen Daten auf sicherem Wege schützt und trotzdem noch die gewohnte Funktionalität bietet. Hierfür muss sichergestellt sein, dass es den Nutzern ohne Fachkenntnis möglich ist, Schlüssel zur Verschlüsselung untereinander auszutauschen und somit festzulegen wer alles diese Positionsdaten erhalten darf. Des weiteren muss beim Schlüsselaustausch gewährleistet sein, dass der Schlüssel auf eine sichere Art und Weise übertragen wird, ohne das andere diesen abfangen können. Hierbei muss allerdings beachtet werden, dass diese Software auf mobilen Geräten lauffähig sein soll. Es sollte also gewährleistet werden dass die genutzten Algorithmen nicht zu berechnungsintensiv sind, die Datensicherheit aber trotz allem gegeben ist.

Da auch ein zentraler Knoten, über den der gesamte Datenverkehr aller Benutzer läuft, unerwünscht ist, muss hier ein Kommunikationsdienst genutzt werden, welcher nach ein dezentrales Prinzip verfolgt. Auch hier muss, wie bei der Datenverschlüsselung, gewährleistet sein dass die Bedienung für den Anwender möglichst einfach gehalten wird und er somit ohne Aufwand und Fachwissen die Kommunikationsparameter einstellen und abändern kann.

Da mittlerweile eine große Anzahl an unterschiedlichen Plattformen für mobile Geräte existieren, sollte gewährleistet sein dass die Software auf möglichst vielen dieser Betriebssystemen lauffähig ist. So ist sichergestellt, dass möglichst viele Benutzer erreicht werden.

2.3 Anwendungsmöglichkeiten

Wenn nun eine Benutzergruppe dieses Programm nutzen möchte, so sollte sie in der Lage sein mit einfachen Mitteln die nötigen Parameter zu verteilen. Hierzu betrachten wir den möglichen Fall, dass eine Gruppe von Touristen in einer ihnen fremden Stadt unterwegs sind. Diese möchten

nun auf eigene Faust diese Stadt erkunden. Hierzu müssen zuerst geeignete Kartendaten vorhanden sein. Hier sollten, wenn möglich, freie Kartendaten verwendet werden um Nebenkosten zu verringern. Es muss allerdings darauf geachtet werden das diese möglichst immer auf einem aktuellen Stand sind. Wenn so eine Karte vorhanden ist, so kann im nächsten Schritt ein Schlüssel erstellt und an alle Teilnehmer dieser Gruppe verteilt werden. Beim Verteilen stellt sich nun die Frage wie dies ohne großen Aufwand aber mit maximaler Sicherheit möglich ist. Hier könnte nun ein Gruppenmitglied einen oder mehrere Schlüssel erstellen und diese sich in Form eines *2D-Barcodes* ausgeben lassen. Dieser Barcode kann nun von den anderen Nutzern, mit deren Smart Phones, gescannt und wieder in einen Schlüssel, bestehend aus Zeichenketten, umgewandelt werden. Da dies keine Kommunikation zwischen den Geräten erfordert kann kein dritter diese Schlüssel während der Datenübertragung abfangen.

Nun stellt sich allerdings auch die Frage nach der Anzahl der zu verteilenden Schlüssel. Je nach Art der Verschlüsselung reicht ein Schlüssel aus, oder es sind mehrere nötig. Würde die Software eine asymmetrische Verschlüsselung benutzen, so müsste jeder Nutzer einen privaten Schlüssel erstellen und sich, mittels scannen eines Barcodes, den zugehörigen öffentlichen Schlüssel besorgen. Dies hätte zur Folge das man, je nach Gruppe mit der kommuniziert werden soll, immer mehrere Schlüssel besitzen und verwalten müsste. Je nach gewähltem Verfahren kann auch zusätzlich noch ein Zertifikat für diese Schlüssel hinzukommen. Ein symmetrischen Kryptographieverfahren hätte hier den Vorteil, dass man nur einen Schlüssel an alle Benutzer der Gruppe verteilen müsste. Des weitern fällt der Verwaltungsaufwand hier wesentlich geringer aus, da nur ein Schlüssel pro Gruppe vonnöten ist. Zusätzlich würde auch noch für ein symmetrisches Verfahren sprechen, dass diese einen wesentlich geringeren Berechnungsaufwand haben und somit der Akku länger halten würde.

Nachdem diese Gruppe von Touristen nun ihre Schlüssel ausgetauscht haben erkunden sie unabhängig voneinander die Stadt. Dabei möchten sie sehen wo die anderen Mitreisenden sich gerade befinden. Allerdings ist es für die Teilnehmer hier nicht von Interesse, die anderen Teilnehmer ausserhalb eines bestimmten Radiuses zu sehen, da der Fussweg zu deren Standort zu weit wäre. Es wäre also nützlich für den Benutzer wenn er ab einem bestimmten Abstand die anderen Teilnehmer ausblenden könnte. Entdeckt nun ein Mitglied dieser Reisegruppe eine besondere Sehenswürdigkeit, möchte er dies vielleicht einem Freund aus der Reisegruppe mitteilen. Zu diesem Zweck bräuchte er eine Chatfunktion, mit welcher er andere Gruppenmitglieder kontaktieren und mit ihnen Nachrichten austauschen kann. Allerdings sollte auch diese Funktion aus Gründen der Privatsphäre, Daten nur in einem verschlüsselten Format übertragen. Des weitern sollte es möglich sein nur mit dieser einen Person gezielt zu chatten, ohne das die gesamte Gruppe mithören kann. Natürlich wäre hier noch eine Zusatzfunktion möglich, die einen Chat mit der gesamten Gruppe verfügbar macht. Wenn dieser Tourist nun seinen Freund per Chat kontaktiert hat, können sich diese nun einen Treffpunkt verabreden und diesen auf der Karte dem anderen Teilnehmer senden und im Anschluss den Weg dorthin sich errechnen lassen.

Ein weiterer möglicher Fall wäre dass man mit Freunden kommunizieren möchte, aber keine *UMTS* Datenflatrate besitzt und kein öffentliches *WLAN* in Reichweite ist. Es sollte also auch sichergestellt sein dass die Datenpakete, welche versandt werden, nicht allzu groß sind. Auch

die Anzahl der Hintergrunddaten, die vom verwendeten Protokoll versandt werden, sollten nicht zu groß ist. So ist zum einen gesichert das nicht zu hohe Verbindungskosten entstehen und zum anderen auch der Berechnungsaufwand für das erstellen und versenden dieser Pakete nicht zu groß ist.

2.4 Anforderungen

Anhand der zwei vorangegangenen Beispielen ist gut erkennbar was an Funktionalität benötigt wird. So sollte es möglich sein die Standorte anderer Nutzer zu auf einer Karte zu anzuzeigen. Damit für andere Nutzer die eigene Position sichtbar ist, muss diese in einem gängigen Format versendet werden. Hierfür ist das Format *Latitude/Longitude* bestens geeignet. Um die Datensicherheit zu garantieren müssen diese Positionsdaten in verschlüsselter Form versendet werden.

Um die Position anderer Teilnehmer zu visualisieren muss das Programm in der Lage sein, die eingehenden Positionsdaten sowohl zu entschlüsseln, als auch diese auf einer Karte darzustellen. Des weiteren muss ein Format für die Karte genutzt werden, welches auf dem mobilen Gerät darstellbar ist und man einfach auf den neusten Stand bringen kann. Es sollte auch möglich sein, nur Benutzer innerhalb einer bestimmten Entfernung anzuzeigen, da eine Person die sich in 6 Kilometer Entfernung aufhält für Dienste dieser Art nur begrenzt sinnvoll sind.

Um die Kommunikation zwischen verschiedenen Teilnehmern zu ermöglichen sollte es möglich sein Chatnachrichten auszutauschen. Auch hier muss gewährleistet sein das der Datenverkehr verschlüsselt ablaufen muss und somit Dritte nicht die Konversation mitlesen können.

Um einen Schlüssel an eine Person weiterzugeben, deren Positon man sehen oder mit ihr kommunizieren möchte, muss es eine Möglichkeit geben diesen Schlüssel auf einfach Weise weiterzugeben. Hierzu kann aus einer vorher festgelegten Zeichenkette einen 2D-Barcode erstellt und angezeigt werden. Zur Weitergabe des Schlüssels muss nun der andere Anwender diesen vom Display fotografieren.

2.5 Verfahren

Anhanden der Anforderungen müssen nun geeignete Verfahren und Protokolle sowohl für Kommunikation als auch für Verschlüsselung gewählt werden. Wie schon erwähnt muss die Verschlüsselung möglichst einfach zu berechnen sein und dabei trotzdem noch bestmögliche Verschlüsselung bieten. Aus Gründen von Berechnungsaufwand als auch von Schlüsselverwaltung

ist ein symmetrisches Verfahren besser geeignet als ein asymmetrisches. Es ist einfacher einen Schlüssel pro Gruppe zu verwalten als einen privaten, einen öffentlichen sowie unter Umständen noch ein Zertifikat. Die Daten werden also durch Client A ver- und von Client B entschlüsselt. Die Schlüssel können, wie schon beschrieben, durch das Erstellen eines *2D-Barcodes* und das fotografieren von diesem zwischen den Anwendern ausgetauscht werden.

Für die Kommunikation zwischen den einzelnen Teilnehmern ist ein dezentrales Protokoll vonnöten, welches möglichst wenig Daten verschickt, die nichts mit der eigentlichen Kommunikation zu tun haben, das stabil läuft und welches beim Ausfallen eines Knotenpunktes diesen mit einem anderen ersetzen kann.

Eine Möglichkeit wäre hier eine *Peer-to-Peer* Lösung. Allerdings verbietet ein Großteil der *GSM-Provider Peer-to-Peer* Datenverkehr innerhalb ihrer Netze, womit diese Lösung praktisch nicht anwendbar wäre. Ein Vorhandenes Protokoll, welches ein aktives Netzwerk bereitstellt und dieses und auch rege genutzt wird, ist das *IRC-Protokoll* [IRC]. Dieses stellt mehrere Server zur Verfügung, wobei jeder Nutzer auch einen eigenen Server bereitstellen kann. Jeder Server verwaltet mehrere *Channels*. Würde nun ein Server ausfallen, so könnten die Nutzer auf einen anderen *IRC-Server* ausweichen. Des Weiteren sind *IRC-Netzwerke* nicht mit einem zentralen Knotenpunkt organisiert, sondern bestehen aus mehreren Servern. Somit ist auch die Anforderung der Dezentralität gegeben, da Nutzer beliebig zwischen den Servern wechseln können. Das Versenden von Hintergrunddaten, wie zum Beispiel Sitzungsinformationen, ist von der Anwendung auf Clientseite frei auswählbar und skalierbar.

Durch die Wahl dieser Lösung ist also garantiert, dass sowohl die Berechnungszeiten durch ein symmetrisches Verfahren niedrig gehalten werden, der Verwaltungsaufwand für die Schlüssel gering ist und die Verteilung der Schlüssel kann durch die angesprochenen Barcodes erfolgen. Bei der Kommunikation gibt es keinen einzelnen Zentralen Server der den gesamten Datenverkehr einsehen kann. Der Verkehr erfolgt über diesen nur in verschlüsselter Form. Des Weiteren kann jeder beliebige *IRC-Server* gewählt werden. Somit sind die Eingangs erwähnten Punkte, Verschlüsselung und dezentrales Protokoll, hiermit abgedeckt und die Privatsphäre ist für den Anwender, im Unterschied zu den meisten anderen Anwendungen dieser Art, gesichert.

3 Technische Grundlagen

Ein wichtiger Punkt ist auch die Wahl der passenden Plattform. Bei geeigneter Auswahl dieser ist es möglich die Software auf mehrere Betriebssysteme für Smart Phones zu portieren und somit eine mehrfache Implementation zu vermeiden. Es wäre somit auch möglich viele Nutzer zu erreichen und die Kommunikation zwischen einem Besitzer eines *iPhones* sowie dem Besitzer eines *Palm Pre's* sicherzustellen.

Des weiteren ist es auch von Interesse, ob andere Programme und Bibliotheken auf den jeweiligen Systemen ausführbar sind. Grundlegend sind Betriebssysteme interessant, die über einen *Layer* verfügen mit welchem man eine große Anzahl an Bibliotheken und Schnittstellen ansprechen kann. Dieser *Layer* soll also eine Schnittstelle zwischen Anwendungen und Betriebssystem sein. Ein solcher ist der *Portable Operating System Interface for Unix Layer (POSIX Layer)* [POSIX]. Mit diesem *Layer* stehen eine große Menge an aktuellen Bibliotheken, aus der *Open-Source* Gemeinde, zur Verfügung welche auch aktiv weiterentwickelt werden. Anwendungen die auf einem *Linux*-System entwickelt wurden können somit ohne weiteres auf ein anderes, *POSIX* kompatibles System, portiert werden.

Auch die Frage der unterstützten Programmiersprachen stellt sich, da das Programm nicht ständig neu implementiert werden soll, wenn es auf ein neues Gerät portiert wird.

Die Problematik der Plattformwahl aufgrund von vorhandener oder nicht vorhandener Hardware ist im Vergleich nicht allzu groß. So haben mittlerweile die meisten der aktuellen Geräte eine ähnliche Ausstattung was Speicher und Prozessorleistung angeht. Auch erweiterte Features wie GPS oder Lagesensoren sind in den meisten aktuellen Geräten vorhanden oder werden in der nächsten Generation, des jeweiligen Herstellers, vorhanden sein.

3.1 Betriebssysteme für mobile Geräte

Wie schon erwähnt ist die Wahl einer geeigneten Plattform nicht unerheblich. Im folgenden werden fünf Betriebssysteme für mobile Plattformen vorgestellt und auf deren Portierungsmöglichkeiten eingegangen.

3.1.1 Windows Mobile

Der wohl bekannteste Vertreter ist *Windows Mobile*. Die aktuelle Version 6.5 wurde von Microsoft auch *Windows Phone* betitelt. Das gesamte Betriebssystem basiert auf der *Windows Win32 API* und lässt Ähnlichkeiten zu den Desktop-Varianten der Windows-Familie erkennen. *Windows Phone* besitzt keinen *POSIX Layer*, allerdings existiert ein *Cross-Compiler* names

CeGCC [CeGCC], mit welchem Programme die in *C/C++* geschrieben wurden für diese Plattform kompiliert werden können.

3.1.2 Android

Das von *Google* entwickelte *Android* [Android] setzt auf einen Linux-Kernel der Version 2.6 auf. Dieser Kernel kümmert sich um die Prozess- und Speicherverwaltung, Kommunikation sowie um die Hardwareabstraktion. Auf diese Grundlage setzt ein virtuelle Java-Maschine auf, in welcher *Android* läuft.

Zum Implementieren von Anwendungen stellt *Google* eigens ein *SDK* bereit. Dieses greift allerdings nur auf *Java*-Bibliotheken zurück, womit sich die nutzbaren Sprachen im Moment eben auf diese beschränken. Des weiteren bietet *Google* mittlerweile ein *NDK* an, mit dessen Hilfe es auch möglich ist Programme in *C* oder *C++* zu schreiben. In diesem Paket werden auch eine Hand voll Bibliotheken mitgeliefert, welchen stabil laufen. *Google* rät allerdings von der Nutzung anderer Bibliotheken ab, da nur die mit dem *NDK* gelieferten stabil auf den Geräten sind. Allerdings ergeben sich hier für die Zukunft, sobald mehr Bibliotheken unterstützt werden, sicher interessante Möglichkeiten für Anwendungsentwicklung und Portierung.

3.1.3 WebOS

WebOS [WebOS] wurde von *Palm* als Nachfolger von *PalmOS* entwickelt und ist momentan nur auf zwei Geräten zu finden: Auf dem *Palm Pre* und dem *Palm Pixi*.

Für dieses Betriebssystem existiert sowohl ein *SDK* für *HTML5*, *CSS* und *Java* sowie ein weiteres, welches im März 2010 veröffentlicht wird, für *C* und *C++*. Des weiteren beinhaltet eine Erweiterung des *POSIX Layers* names *PIPS*. Somit werden mehrere Programmiersprachen unterstützt und es besteht die Möglichkeit den *POSIX Layer* zu nutzen.

3.1.4 iPhone OS

Bei *iPhoneOS* [iPhoneOS] handelt es sich um eine abgeänderte und angepasste Version von *MacOS*. Es wurde eigens für das iPhone entwickelt. Auch für dieses System existiert ein *SDK*, welches allerdings nur die Sprache *Objective-C* unterstützt. Des weiteren fehlt auch eine Unterstützung des *POSIX Layers*. Der größte Kritikpunkt an diesem System dürfte allerdings das

fehlen von *Multitasking*-Unterstützung sein. Somit ist es nicht möglich zwei Anwendungen parallel auszuführen, was gerade *location awareness* Anwendungen stark einschränkt, da hier häufig weitere Dienste im Hintergrund aktiv sein sollten.

3.1.5 Symbian OS

SymbianOS [SymbianOS] ist ein Betriebssystem welches vorzugsweise auf Geräten der Firma *Nokia* zum Einsatz kommt. Es existiert ein *SDK*, was neben *C/C++* auch noch weitere Sprachen wie zum Beispiel *Python* oder *Java* unterstützt. Mit dem *SDK* wird auch ein *Cross-Compiler* angeboten, welcher es ermöglicht Programme direkt zu portieren. Des Weiteren besitzt *Symbian OS* auch einen *POSIX Layer*.

3.1.6 Zielplattform

Die Wahl der Zielplattform ist auf *Windows Mobile* gefallen, da es hier möglich ist das Programm in *C* zu schreiben und dann im Anschluss nach *Windows Mobile* zu portieren.

iPhoneOS wurde aufgrund seiner mangelnden *Multitasking*-Unterstützung ausgeschlossen. Diese ist für den geplanten Dienst wichtig, da hier Prozesse im Hintergrund stattfinden werden und dies auf einem solchen System nicht realisierbar wäre. *Android* hat zwar eine *C* Unterstützung, allerdings gibt der Hersteller an das nicht alle Bibliotheken stabil sind.

Aufgrund der Implementierung in *C* ist es auch möglich das Programm für *WebOS* und *SymbianOS* zu kompilieren.

3.2 Softwaregrundlagen

Aufgrund der gewählten Zielplattform und Programmiersprache muss nun eine Möglichkeit gefunden werden das Programm sowohl für die jeweiligen Plattformen zu kompilieren, sowie die graphischen Elemente auf den Plattformen darzustellen.

3.2.1 CeGCC

Da *Windows Mobile* und die Programmiersprache *C* genutzt wird, wird der *CeGCC* als *Cross-Compiler* verwendet. Mit ihm ist es möglich *C* Programmcode, der unter *Linux* entwickelt wurde nach *Windows Mobile* zu portieren. Bei *CeGCC* handelt es sich um ein *Open-Source* Projekt, basierend auf dem *GCC*. Mit diesem Tool können in einer *Linux* Umgebung die für *Windows*

Mobile benötigten Bibliotheken und ausführbaren Dateien erstellt werden.

Es wird zwischen zwei verschiedenen Arten des *CeGCC*'s unterschieden. Zum Einen *CeGCC*, zum Anderen *mingw32ce*. Der Unterschied zwischen diesen beiden Kompilern besteht darin, dass ersterer nur dann benutzt wird, wenn man nur Linux Bibliotheken einbinden möchte. Der *mingw32ce*-Kompiler wird dann gebraucht, wenn man auch *Windows Mobile* Bibliotheken nutzt.

Soll das Programm nun für *WebOS* oder *SymbianOS* portiert werden, kann dies auf unter *Linux* normal kompiliert werden.

3.2.2 Enlightenment

Neben einem *Cross-Compiler* wird noch ein geeignetes Frontend benötigt, um das Programm auch für den Benutzer ansprechend darzustellen sowie eine einfache Bedienbarkeit zu garantieren. Dieses Frontend sollte auch in *C* oder *C++* geschrieben sein um auch hier die Portierbarkeit für die gewünschten Plattformen zu garantieren. Hier fiel die Wahl auf das freie, seit 1997 existierende, *Enlightenment* [efl] Projekt. Dieses Softwarepaket unterstützt alle gängigen Plattformen wie Windows, Linux, BSD und MacOS. Es beinhaltet einen eigenen *Window-Manager* names *Elementary*. *Elementary* bietet ein umfangreiches Paket an grafischen Elementen die genutzt und frei angeordnet werden können.



Abbildung 1: Beispiele verschiedener *Elementary* Icons

Elementary setzt auf die *Enlightenment Foundation Libraries (EFL)* auf. Diese Bibliotheken werden zum Teil von Enlightenment benötigt, andere können für optionale Features installiert werden. Für die Darstellung auf mobilen Geräten sind die Pakete *Evil*, *Eina*, *Eet*, *Embryo*, *Evas*, *Ecore*, *Edje* und *Elementary* nötig.

Bei *Ecore* handelt es sich um eine *library* welche das serialisieren von mehreren Programmteilen ermöglicht und für den Betrieb auf mobilen Geräten optimiert wurde. *Edje* ist eine komplexe

Abbildung 2: Aufbau von *Enlightenment*

grafische Design und Layout Bibliothek, welche mit einer internen *state machine* und einem Zustandsgraphen speichert was wo, in welcher Farbe und wie sichtbar ist und gezeichnet werden soll. Die Bibliothek *Evas* ist eine *canvas*-Bibliothek, welche sich um Effekte wie Alpha-Blending oder das skalieren von Bildern kümmert. *Eina* stellt verschiedene, optimierte Datentypen und Tools bereit.

Im Anhang 1 sind genaue Anweisungen zu finden, um *Enlightenment* für *Windows Mobile* zu erstellen.

4 Friend Finder

Die eingangs beschriebene Software hat den Namen *Friend Finder* und wurde im Rahmen dieser Arbeit mit fast allen aufgezählten Funktionen realisiert. Das fotografieren des Barcodes, der Gruppenchat sowie das Setzen von Marken auf der Kartesind in der Implementation nicht enthalten. Im folgenden wird auf die Verwendeten Verfahren sowie Bibliotheken, die zur Realisierung notwendig waren, eingegangen.

4.1 Verwendete Verfahren und Bibliotheken

Friend Finder wurde so konzipiert, dass die Graphische Darstellung ohne großen Aufwand vom den restlichen Teilen der Software abgekoppelt und durch eine andere, darstellende Bibliothek ersetzt werden kann. Somit könnte man *Enlightenment* durch eine andere Art der Darstellung austauschen, ohne dabei die Funktionalität der zugrunde liegenden Komponenten zu zerstören. Da das Ver- und Entschlüsseln der Daten möglichst wenig Rechenaufwand erzeugen und der Schlüsselaustausch nicht zu kompliziert sein soll, nutzt das Programm ein symmetrisches Verschlüsselungsverfahren.

Abbildung 3 zeigt den Kommunikationsaustausch von *Friend Finder*. Der *Message Sender* ist für das Versenden und Empfangen der Textnachrichten zuständig, *Sender* sendet die eigene Position, *Receiver* empfängt die Positionen der anderen Nutzer und sendet Acknowledgements an die teilnehmenden *Sender*. Alle drei Teile geben ihre empfangenen Daten an die *GUI* weiter, welche sie mit Hilfe von *Enlightenment* ausgibt.

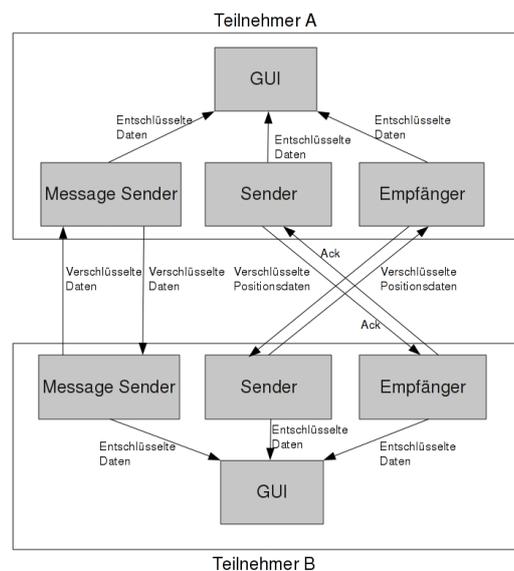


Abbildung 3: *Friend Finder* Nachrichtenaustausch

4.1.1 Grafisches Benutzeroberfläche

Zum Erstellen der Oberfläche wurde *Enlightenment* verwendet. Diese Bibliothek stellt alle benötigten Funktionen bereit und bietet eine Fülle an vordefinierten Bedienelement. Der gesamte Programmcode der Benutzeroberfläche wurde in einer Datei zusammengefasst (*gui.c*). Diese Tatsache vereinfacht das Erhalten der Modularität, da einfach nur diese Datei durch eine andere ersetzt werden muss um einen anderen Typ von Oberfläche zu benutzen.

In der der Datei *gui.c* sind alle Funktionen enthalten um die Oberflächenelement zu erzeugen und zu platzieren. Um die gewünschte Funktionalität der einzelnen Elemente zu realisieren wurden auch die Aufrufe der benötigten Funktionen aus anderen Modulen in dieser Datei implementiert. Wie schon erwähnt, wurde die Graphische Nutzeroberfläche mit Hilfe von Bibliotheken aus dem *Elementary*-Paket realisiert. Zur Darstellung der Karte wurden Daten des offenen Kartenprojekts *OpenStreetMap* [OSM] genutzt.

4.1.2 Versenden der Nachrichten

Um Daten im Allgemeinen zu versenden wurde das *IRC*-Protokoll verwendet. Die Vorteile dieses Protokolles liegen in seiner weiten Verbreitung, einer ausgedehnten Serverstruktur, sowie in dessen Stabilität.

In der Datei *msg_sender.c* sind alle Funktionen und Aufrufe implementiert, welche nötig sind um die Verbindung zum *IRC-Server* zu erstellen und die Nachrichten zu verschicken. Um eine Verbindung zu einem gegebenen *IRC-Server* zu erstellen muss eine *IRC-Session* initialisiert werden. Diese *Session* beinhaltet Informationen wie zum Beispiel den *Nickname* des Benutzers oder die *IP-Adresse* des Servers. Nachdem diese *Session* gestartet wurde, kann man nun durch das Aufrufen der Funktion "*set_txt_msg(char* msg)*" die Nachricht versenden. Wird eine Nachricht empfangen so wird diese an die Funktion "*show_message(char* msg)*", welche zur Benutzeroberfläche gehört, übergeben. Bei der Implementierung des Nachrichtenversandes ist eine Besonderheit zu erwähnen. Das genutzte Verschlüsselungsverfahren *Blowfish* wurde seitens der *OpenSSL*[OpenSSL] Bibliothek als *Blockcipher* implementiert. Das bedeutet, dass immer nur maximal 64 Bit Nachrichten verschlüsselt werden können. Da in der Programmiersprache *C* dies genau acht ASCII-Zeichen entspricht, werden alle zu sendenden Nachrichten in Blöcke der Größe acht aufgeteilt, versandt und beim Empfänger wieder zusammengesetzt.

Ein weiterer wichtiger Unterschied zu den Modulen Senden und Empfangen von *GPS*-Positionen ist die Tatsache, dass bei diesem Programmteil Sender und Empfänger in der gleichen Datei implementiert wurden. Der Grund hierfür ist, dass man hier nicht zwischen mehreren Sendern oder Empfängern unterscheiden muss, und diese zwei Teile hier somit nicht komplett getrennt voneinander arbeiten müssen. In der folgenden Abbildung ist eine Konversation über *Friend Finder* zu sehen.

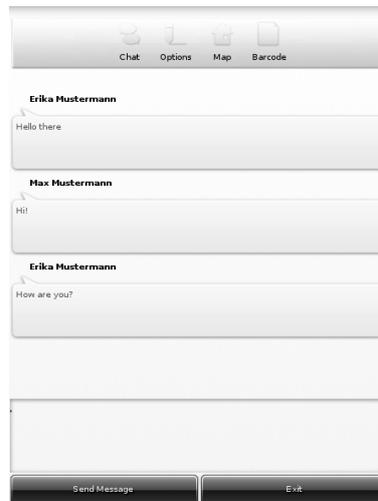


Abbildung 4: Versenden von Chatnachrichten

Um Nachrichten zu versenden wurde für dieses Projekt die *IRC-Client Bibliothek*[libircclient] verwendet. Diese *library* bietet verschiedene Funktionen um eine Verbindung mit einem *IRC-Server* zu erstellen und Nachrichten an diesen zu senden, sowie eingehende Nachrichten zu empfangen.

Zur Ver- und Entschlüsselung der gesendeten Nachrichten, sowie der Positionsdaten wird die Bibliothek des *OpenSSL-Projekts*[OpenSSL], namens *libcrypto*, verwendet. Hierzu werden die Daten mit dem *Blowfish-Algorithmus* verschlüsselt. Bei diesem Algorithmus handelt es sich um ein symmetrisches Verfahren, bei welchem alle Teilnehmer den gleichen privaten Schlüssel zum ver- sowie entschlüsseln nutzen.

4.1.3 Versenden der eigenen Position

Der benötigte Programmcode zum Versenden der eigenen Position ist in der Datei *sender.c* zu finden. Auch hier muss zuerst eine *IRC-Session* initialisiert werden um danach die Position zu versenden. Der Ablauf beim Senden der Positionen erfolgt in einer vorgegebenen Reihenfolge. Zuerst wird der verschlüsselte Längengrad, danach der verschlüsselte Breitengrad gesendet. Allerdings muss auch hier, wie beim Versenden der Textnachrichten, darauf geachtet werden dass maximal eine Zeichenkette der Länge 8 Byte verschlüsselt wird. Somit ist es auch hier nötig Längen- und Breitengrad in zwei Teile aufzuteilen und getrennt zu versenden. Somit werden für das Versenden einer Position insgesamt acht Nachrichten an den *IRC-Server* übermittelt. Wurden diese vier Nachrichten übermittelt, so werden solange keine Daten mehr gesendet, bis der Empfänger eine Bestätigung an den *IRC-Kanal* sendet. Diese Bestätigung ist wird unverschlüsselt versandt. Kommt dieses *Acknowledgement* beim Sender an, so versendet dieser wieder ein *Latitude/Longitude Paar*.

Auch hier wird, wie beim Versenden der Nachrichten zum Verschlüsseln der *Blowfish-Algorithmus* aus *libcrypto*, sowie *libircclient* zum versenden der Daten genutzt.

4.1.4 Empfangen der eigenen Position

Das Verhalten des Empfängers beim Erhalten einer Nachricht ist etwas komplizierter. Im ersten Schritt muss auch hier eine *IRC-Session* initialisiert werden. Da mehrere Benutzer Positionsdaten senden können legt der Empfänger für jeden Sender einen Datensatz an. Dieser wird nach und nach mit den Positionsdaten gefüllt und die benötigten Daten weitergegeben, sobald alle vorhanden sind. Da der Sender seine Daten, aufgrund der Restrektion der Länge der zu verschlüsselnden Zeichenkette, gestückelt sendet ist es von Nöten das der Empfänger die Daten dem jeweiligen Absender zuordnen kann und diese auch wieder korrekt zusammensetzt. Dies geschieht mit Hilfe von Terminierungszeichen am Ende eines jeden Positionsbruchstückes. Dieses Zeichen wurde vom Sender angehängt und der Empfänger kann mit dieser Hilfe erkennen wie die gesendeten Daten zugeordnet werden müssen. Wenn dies geschehen ist und sowohl *Latitude* als auch *Longitude* Informationen vorhanden sind, werden diese Position an das Frontend weitergegeben.

Zur Realisierung des Empfängers werden die gleichen Bibliotheken wie beim Sender genutzt. Auch hier wird zur Entschlüsselung der *Blowfish-Algorithmus* von *libcrypto* genutzt.

4.1.5 Erzeugen eines 2D-Barcodes

Die Datei *barcode.c* beinhaltet die Funktionen zum Erstellen eines 2D-Barcodes. Hierzu wird die Funktion "*generate_barcode(char* key)*", mit einer Zeichenkette als Übergabeparameter, aufgerufen. Aus dieser Zeichenkette wird dann ein Barcode erstellt, welcher im darauf folgenden Schritt als *.png* Datei auf das Speichermedium geschrieben wird. Zum erstellen des Barcodes wurde die offene Bibliothek *qrencode* [qrencode] genutzt. Diese erstellt aus einer Zeichenkette einen 2D-Barcode. Aus den Bilddaten dieses Barcodes wurde mit *libpng* [PNG] eine *.png* Datei erstellt. Die untere Abbildung zeigt einen solchen erstellten Barcode, wie er von *Friend Finder* ausgegeben wird.

4.2 Analyse

Das Ziel war es mit *Friend Finder* Daten verschlüsselt zu übertragen. Es soll dabei ein möglichst geringer Berechnungsaufwand entstehen um die Daten zu verschlüsseln, sowie möglichst wenig Datenoverhead produziert und versendet werden. Unter Datenoverhead werden die Hintergrunddaten gesehen, welche versendet werden um zum Beispiel die Verbindung zwischen Client und Server aufrecht zu erhalten oder um zu kontrollieren ob Server oder Client noch verfügbar



Abbildung 5: 2D-Barcode mit *Friend Finder*

sind. Diese Daten sind von Interesse da mit vielen versendeten Daten ein höherer Anspruch des Rechenkerns einhergeht, was wiederum in einem höheren Stromverbrauch resultiert.

Im folgenden Teil wird der erzeugte Datenverkehr von *Friend Finder* analysiert. Ein Hauptaugenmerk wird hierbei vor allem auf die Paketgröße, sowie die Menge der versendeten Datenpakete geworfen. Der *Traffic* wurde mit Hilfe des Programmes *Wireshark* [Wireshark] untersucht. Wie bereits erwähnt wird zum Versenden der Nachrichten das *IRC-Protokoll* verwendet. In dieser Testumgebung wurde die Software *IRCD-Hybrid* [IRCD] genutzt. Der Server lief auf dem gleichen Computer wie der Client und der Client hat sich über das *localhost* Interface mit dem Server verbunden.

Die Analyse ist in drei Teile aufgeteilt. Als erstes wird auf den allgemein entstehenden Datenverkehr eingegangen, welcher bei Verbindungsaufbau, sowie bei Beenden der Verbindung entsteht. Der zweite Teil beschäftigt sich mit dem Versenden sowie Empfangen von Nachrichten. Im letzten Teil dieser Analyse wird auch das dritte Feature, dass Versenden und Empfangen von Positionen, unter die Lupe genommen.

4.2.1 Allgemeiner Datenverkehr

Bei Messung des allgemeinen Datenaufkommens mit einer normalen *IRC-Clients*, hier *X-Chat* [xchat], ergeben sich folgender Datenverkehr.

Beim Verbindungsaufbau sendet der *IRC-Server* zuerst ein Paket mit Informationen wie Limit der *Channels* oder Anzahl der aktiven Benutzer. Dieses Paket hatte in der Versuchsumgebung

eine Größe von 1090 Bytes. Hiervon müssen noch 20 Bytes *IP*- und 32 Byte *TCP-Header* abgezogen werden. Somit sind die hat das Datenfeld eine Größe von 1038 Bytes. Bei im folgenden genannten Paketgrößen sind diese 52 Byte schon abgezogen.

Wenn im nächsten Schritt der Benutzer nun eine *Channel* beitrifft so sendet dieser drei Pakete an den Server. Diese beinhalten den Namen des *Channels* dem beigetreten werden soll, eine Anfrage der aktiven Nutzer in diesem *Channel* sowie welche Rechte der beitretende Nutzer in diesem *Channel* inne hat. Diese drei Pakete haben alle die Größe von 26 Byte. Ist die Verbindung zwischen Client und Server aufgebaut so sendet der Client alle 30 Sekunden ein *Ping* Paket an den Server, welches dieser mit einem *Pong* beantwortet. Diese Pakete haben eine Größe von 40 Byte für die *Ping* Nachricht und 58 Byte für die beantwortende *Pong* Nachricht. Alle 60 Sekunden versendet der Client eine Anfrage, welche Teilnehmer sich im *Channel* befinden. Diese Nachricht von Client zu Server ist 25 Byte groß. Die Antwort hierzu ist abhängig von der Anzahl der Benutzer. Ist nur ein Benutzer im *Channel* so ist sie 151 Bytes groß, bei zwei ist sie schon 233 Byte groß. Wird eine Verbindung beendet, so schickt der Client noch eine *Quit* Nachricht an den Server.

Im Gegensatz zu diesem hohen Hintergrundverkehr benötigt *Friend Finder* nur einen *TCP-Handshake* um die Verbindung aufzubauen. Ist die Verbindung etabliert fallen keine weiteren Hintergrunddaten an. Der Nachteil hiervon ist, dass der Client nur bei einem auftretenden Fehler beim versenden der Daten bemerkt das er nicht mehr verbunden ist. Allerdings wird hiermit auch Berechnungszeit und somit Akku gespart, da diese *Keep-Alive* Nachrichten, sowie die zusätzliche Informationen über den Server im Rahmen von *Friend Finder* nicht benötigt werden, da die Positionsdaten mehrmals pro Minute übermittelt werden und somit immer Datenverkehr zwischen Sender und *IRC*-Server vorhanden ist.

4.2.2 Versenden und Empfangen von Nachrichten

Um das Versenden von Nachrichten zu evaluieren wurde "Hello World" als Testnachricht benutzt. Der *Blockcipher* von *Friend Finder* teilt den Satz "Hello World" in zwei Teile auf: "Hello " und "World". Diese werden dann von *TCP* aufgrund der Fenstergröße in ein Paket gepackt. Das gesamte Paket hat die Größe von 147 Bytes, wobei *TCP*- und *IP-Header* abgezogen werden müssen. Somit haben die Daten eine Größe von insgesamt 81 Bytes.

Beachtet man dass ein *char* in *C* die Größe von einem Byte hat und der Beispielsatz aus elf Zeichen besteht, so ist dieser unverschlüsselt 11 Byte groß. Nach der Verschlüsselung werden beim Senden noch Informationen wie der *Channel* und der Empfänger der Nachricht in das zu versendende *IRC*-Paket geschrieben. Somit vergrößert sich eine Textnachricht circa um den Faktor 7,4, wenn sie verschlüsselt ist und alle Zusatzinformationen hineingepackt wurden.

Wenn h die Größe des *TCP-Headers* und t die Anzahl der Zeichen der unverschlüsselten Nachricht ist, so ergibt sie die ungefähre Größe der zu versendenden Nachricht aus: $h + (t \cdot 7,4)$.

4.2.3 Versenden und Empfangen von Positionen

Wie schon erwähnt, werden die Positionsdaten beim Sender aufgeteilt und mit vier unterschiedlichen Nachrichten versandt. Wie beim Versenden der Textnachrichten werden diese vier Nachrichten auch hier in ein Paket gepackt. Dieses hat die Gesamtgröße von 235 Byte. Abzüglich der *TCP*- und *IP-Header* ergibt sich hier eine Datengröße von 169 Byte. Die Anzahl der unverschlüsselten Zeichen, die zu senden sind, beträgt 21 Zeichen. Diese sind jeweils 1 Byte groß, womit sie in der Summe also 21 Byte Größe haben. Durch die Verschlüsselung sowie die benötigten Zusatzinformationen, wie *Channel* für den die Nachricht bestimmt ist, vergrößert sich das Datenvolumen also um circa den Faktor acht. Wenn h die Größe des *TCP-Headers* und t die Anzahl der Zeichen der unverschlüsselten Nachricht. Somit ergibt sich die Größe der versendeten Nachricht circa durch $h + (t \cdot 8)$. Hinzu kommt noch, das für jedes empfangene Positions-Fragment ein Acknowledgement gesendet wird. Somit sind nach Empfangen aller vier Positionsteile vier Acknowledgements der Größe von 124 Bytes insgesamt und 58 Bytes an Daten ohne Header gesendet worden. Somit kann aus Folgende Formel für den Datenverkehr über die Zeit hergeleitet werden: $((h + (t \cdot 8)) + (4 \cdot a)) \cdot n$, wobei a die Größe eines Acknowledgement-Paketes ist und n die Anzahl der versandten Pakete repräsentiert.

4.2.4 Fazit der Auswertung

Im Bereich des Allgemeinen Datenverkehrs fällt kein Overhead durch *Friend Finder* an. Hier werden nur dann Daten versandt, wenn dies auch vom Nutzer so gewollt sind und auch gebraucht werden. Dies hat den Vorteil, dass der Anwender die totale Kontrolle über die Daten, die er sendet, hat. Allerdings kann so auch erst beim Versenden von Daten festgestellt werden, ob die Verbindung unterbrochen wurde, da im Hintergrund die Verbindung nicht ständig überprüft wird. Aufgrund der Tatsache, dass Positionsdaten mehrmals pro Minute versandt werden, ist dies aber auch nicht wirklich nötig da durch das Versenden oder nicht Versenden dieser Daten auch ein Verbindungsproblem festgestellt werden könnte

Ein weiterer Vorteil von *Friend Finder* ist das Verschicken von Daten an mehrere Benutzer. Hier muss der Client sein Paket nur einmal versenden und alle teilnehmenden Nutzer in einem Channel können diese Nachricht einsehen, sofern dies gewollt ist. Würde man hier eine Verbindung pro Teilnehmer nutzen, so müsste man für n Teilnehmer n Verbindungen öffnen und auch n Mal die Daten versenden. Dies würde ein nicht unerheblicher Berechnungsaufwand sowie Datenverkehr darstellen. Im Empfangen der Daten ergibt zwischen der Lösung durch *Friend Finder* oder einer Verbindung pro Teilnehmer keinen Unterschied, da bei beiden Modellen diese Daten nur einmal empfangen werden.

5 Fazit

Anhand der Anforderungen und der implementierten Features von *Friend Finder* ist ersichtbar das es möglich ist einen sicheren *location awareness* Dienst zu entwerfen und programmieren. Dieser verschlüsselt die versendeten Nachrichten und verzichtet auf einen zentralisierten Dienst ohne dabei an Verlässlichkeit einzubüßen. Auch das Datenaufkommen, sowie die Berechnungszeit der Verschlüsselung kann bei richtiger Implementierung gering gehalten werden. Somit wird auch der Akku von mobilen Geräten geschont.

Es ist also möglich solche Dienste, die die Privatsphäre eines Nutzers betreffen, auch unter dem Aspekt der Sicherheit zu implementieren und somit diese zu Garantieren. Die gegebenen Möglichkeiten eines Missbrauchs der versandten Daten sind somit stark eingeschränkt und können nicht mehr ohne weiteres eingesehen werden. Des weiteren können mit Hilfe von 2D-Barcodes die, zur Verschlüsselung benötigten, Schlüssel einfach und ohne Risiko ausgetauscht werden.

Bei richtiger Wahl der Programmiersprache sowie der Schnittstellen und genutzten Bibliotheken kann man des weiteren eine größere Menge von Betriebssystemen für Smart Phones abdecken und somit mehr Nutzer erreichen, ohne dies in unterschiedlichen Sprachen und mit unterschiedlichen Bibliotheken implementieren zu müssen. Gerade diese und die Tatsache das die Betriebssysteme für mobile Geräte immer mehr die gleichen Schnittstellen benutzen, zum Beispiel den *POSIX-Layer* könnte es in der Zukunft ermöglichen Programme noch einfacher zu portieren und den Aufwand bei Entwurf und Design geringer zu halten.

Anhang

Anhang 1

Um die Pakete *Evil*, *Eina*, *Eet*, *Embryo*, *Evas*, *Ecore*, *Edje* und *Elementary* von Ubuntu nach Windows Mobile zu portieren, sind die folgenden Schritte nötig:

Im allerersten Schritt muss eine aktuelle Version des *CeGCC*'s heruntergeladen und installiert werden. Die benötigten Dateien können auf der Projekthomepage gefunden und heruntergeladen werden. Für diese Aufgabe wird der *mingw32ce* für den *ARM*-Prozessortyp benötigt. Dieser muss auf der Homepage des CeGCC-Projekts heruntergeladen und in das passende Systemverzeichnis entpackt werden.

Bevor man mit dem nächsten Schritt fortfahren kann, müssen noch ein paar benötigte Pakete aus dem Ubuntu-Repository installiert werden.

```
sudo apt-get install build-essential make gcc bison flex subversion
autoconf libtool gettext libfreetype6-dev libpng12-dev zlib1g-dev
libjpeg-dev libtiff-dev libungif4-dev librsvg2-dev xorg-dev
libltdl3-dev libcurl4-dev cvs subversion git-core doxygen proj
libsqlite3-0 libsqlite3-dev
```

Nachdem diese Pakete installiert wurden kann man sich nun die einzelnen Pakete aus dem *Subversion-Repository* der Entwickler herunterladen.

Nun muss man sich noch ein Verzeichnis anlegen, in welchem die für Windows Mobile kompilierten Dateien abgelegt werden. Des weiteren muss noch eine Datei angelegt werden, in welcher die Pfade zu dem genutzten Compiler liegen und welche dann einmalig exportiert werden müssen, damit die benötigten *Header-Files*, *textitLibraries* und *Binaries* auch vom Betriebssystem gefunden werden. Diese Datei wird im folgenden "mingw32ce.env" benannt.

```
touch mingw32ce.env
```

Nun müssen noch in diese Datei die zu exportierenden Pfade geschrieben werden.

```
export CEGCC_PATH=/opt/cegcc
export MINGW32CE_PATH=/opt/mingw32ce
export WINCE_PATH=$HOME/workspace/wince
```

```
export PATH=$CEGCC_PATH/bin:$MINGW32CE_PATH/bin:$PATH
export CPPFLAGS="-I$WINCE_PATH/include -I$WINCE_PATH/zlib-1.2.3-dev/include
-I$WINCE_PATH/libjpeg-6b-dev/include -I$WINCE_PATH/win_iconv-dev/include
-I$WINCE_PATH/freetype-2.3.7-dev/include
-I$WINCE_PATH/libpng-1.2.33-dev/include/libpng12
-I$WINCE_PATH/win_iconv-dev/include -I/opt/mingw32ce/arm-mingw32ce/include/"
export LDFLAGS="-L$WINCE_PATH/lib -L$WINCE_PATH/zlib-1.2.3-dev/lib
-L$WINCE_PATH/libjpeg-6b-dev/lib
-L$WINCE_PATH/win_iconv-dev/include -L$WINCE_PATH/freetype-2.3.7-dev/lib
-L$WINCE_PATH/libpng-1.2.33-dev/lib -L$WINCE_PATH/win_iconv-dev/lib
-L$CEGCC_PATH/lib"
export LD_LIBRARY_PATH="$WINCE_PATH/bin"
export PKG_CONFIG_PATH="$WINCE_PATH/lib/pkgconfig"
```

Der Inhalt dieser Datei muss nun in jeder neu geöffneten Shell neu exportiert werden, da die Variablen durch die hier gewählte Methode nur in diesen Shell's existieren, in denen sie exportiert wurden.

Bei den Variablen "CEGCC_PATH" und "MINGW32CE_PATH" ist der Pfad zum Verzeichniss des *cegcc*, beziehungsweise des *mingw32ce* Kompilers einzutragen. Unter "WINCE_PATH" muss der Pfad, zu dem Verzeichniss in dem die kompilierten Daten gespeichert werden sollen, eingetragen werden. Mit "PATH" werden die *Binaries*, der zwei Kompiler, in den Systempfad aufgenommen. Des weiteren werden unter "CPPFLAGS" die *include*-Pfade und unter "LDFLAGS" die *Librarie* Pfade abgelegt. "LD_LIBRARY_PATH" zeigt auf den Ordner in welchem die kompilierten *Binaries* liegen. "PKG_CONFIG_PATH" zeigt schliesslich noch auf den Ordner der die Paketinformationen der installierten Dateien beinhaltet. Dieses exportieren geschieht mit dem folgenden Aufruf.

```
source <Pfad-zu-der-Datei>/mingw32ce.env
```

Im nächsten Schritt muss nun noch ein Ordner angelegt werden, in welchem der *Enlightenment Source-Code* abgelegt wird. Nun muss noch in dieses Verzeichniss gewechselt werden und es kann mit dem ersten Programm begonnen werden.

Evil

Als erstes ist es nötig das Programm *Evil* aus dem *SVN*, welches von den Entwicklern bereit gestellt wurde, herunterzuladen. Das Herunterladen geschieht mit:

```
svn co http://svn.enlightenment.org/svn/e/trunk/evil
```

Nachdem alle Dateien erfolgreich heruntergeladen wurden muss, falls nicht schon geschehen, die Datei mit den *Umgebungsvariablen* eingelesen werden. Nachdem dies geschehen ist, kann man nun das Konfigurationsskript starten

```
./autogen.sh --prefix=$WINCE_PATH --host=arm-mingw32ce
```

Durch diesen Aufruf wird der Installationspfad auf den Wert der Variable “WINCE_PATH” gesetzt, als Zielsystem ein *ARM-Prozessor* gewählt und der *mingw32ce*-Kompiler als Kompiler gewählt. Nachdem dieses Skript erfolgreich durchgeführt wurde, kann man im nächsten Schritt das Programm erstellen.

```
make
```

Ist auch dies erfolgreich durchgeführt worden muss man nun noch in einem letzten Schritt die erstellten Dateien im Zielordner installieren.

```
make install
```

Nun sollte *Evil* erfolgreich im Zielordner installiert worden sein.

Eina

Auch hier ist es auch wieder nötig die Dateien aus dem Entwickler-Repository herunterzuladen.

```
svn co http://svn.enlightenment.org/svn/e/trunk/eina
```

Danach wird auch hier wieder das “autogen.sh” Skript aufgerufen.

```
./autogen.sh --prefix=$WINCE_PATH --host=arm-mingw32ce --disable-pthread
```

Es werden bei diesem Aufruf die gleichen Parameter wie bei *Evil* übergeben. Hinzu kommt noch “--disable-pthread”. Mit diesem Parameter wird *threading* beim Erstellen von *Eina* deaktiviert, da *ARM-Prozessoren* dies nicht unterstützen. Nachdem das Skript durchgelaufen ist, muss man nun auch wieder das Programm erstellen und im Zielverzeichnis installieren.

```
make ; make install
```

Eet

Bevor man *Eet* erstellen kann, muss man noch vier vorgefertigte *tar-Archive* im Verzeichniss, welches in der Variable "WINCE_PATH" gespeichert wurde, entpacken. Diese Archive kann man unter den Links, welche in Anhang 2 zu finden sind, herunterladen. Nach dem Herunterladen müssen diese nur noch in das "WINCE_PATH"-Verzeichniss kopiert und entpackt werden. Nun kann man den Quellcode für *Eet* herunterladen.

```
svn co http://svn.enlightenment.org/svn/e/trunk/eet
```

Nachdem die Dateien heruntergeladen sind, muss wieder das "autogen.sh"-Skript aufgerufen werden.

```
./autogen.sh --prefix=$WINCE_PATH --host=arm-mingw32ce
```

Im Anschluss muss nun auch wieder kompiliert und installiert werden.

```
make ; make install
```

Embryo

Der Erste Schritt ist auch hier das Herunterladen des Programmcodes.

```
svn co http://svn.enlightenment.org/svn/e/trunk/embryo
```

Nachdem die Dateien heruntergeladen sind auch wieder das Skript aufrufen.

```
./autogen.sh --prefix=$WINCE_PATH --host=arm-mingw32ce
```

Im Anschluss nun auch wieder kompilieren und installieren.

```
make ; make install
```

Evas

Auch für *Evas* müssen mehrere *tar-Archive* heruntergeladen werden (siehe Anhang 3). Auch sollten diese in das gleiche Verzeichniss, wie die vorhergegangenen Archive, entpackt werden. Nun müssen noch die Dateien, welche die Paketinformationen beinhalten, um die heruntergeladen Pakete ergänzt werden:

```
cp $WINCE_PATH/cp libpng-1.2.33-dev/lib/pkgconfig/libpng*
  $WINCE_PATH/lib/pkgconfig/
cp $WINCE_PATH/freetype-2.3.7-dev/lib/pkgconfig/freetype2.pc
  $WINCE_PATH/lib/freetype2.pc
```

Nun müssen diese Paketinformationen noch bearbeitet werden. Dazu müssen diese mit einem beliebigen Editor geöffnet werden und in beiden Dateien der Wert von "prefix" auf "WINCE_PATH" gesetzt werden.

Nachdem dies durchgeführt wurde kann nun *Evas* heruntergeladen werden.

```
svn co http://svn.enlightenment.org/svn/e/trunk/evas
```

Nun muss auch hier, wie bei allen anderen Programmen das "autogen.sh"-Skript aufgerufen werden.

```
./autogen.sh --prefix=$WINCE_PATH --host=arm-mingw32ce --disable-async-events
```

Als nächster Schritt muss nun das Programm kompiliert werden.

```
make
```

Sollte hierbei die Datei "ft2build.h" nicht gefunden werden, so muss diese an die richtige Stelle kopiert werden. Eigentlich liegt die Datei an folgendem Ort:

```
$WINCE_PATH/freetype-2.3.7-dev/include/freetype2/ft2build.h
```

Allerdings wird sie im Ordner "freetype2" nicht gefunden. Um dies zu umgehen muss "ft2build.h" einfach eine Ordner Ebene nach oben kopiert werden.

```
cp $WINCE_PATH/freetype-2.3.7-dev/include/freetype2/ft2build.h $WINCE_PATH/freetype-
```

Um einen weiteren Fehler von vorneherein zu umgehen, muss man noch den Pfad eines eingebundenen Headers in "ft2build.h" abändern. Hierzu öffnet man "ft2build.h" mit einem beliebigen Editor und ändert folgendes

```
#include <freetype/config/ftheader.h>
```

zu

```
#include <freetype2/freetype/config/ftheader.h>
```

ab. Anschliessend muss nun auch der "freetype"-Ordner um eine Ebene nach oben kopieren werden, da die *include*-Pfade in den Headern von "freetype2" nicht korrekt sind. Falls man nun noch *evas* mit *DirectX-Support* kompilieren möchte, muss man das *DirectX-SDK* herunterladen und "ddraw.h" in die Verzeichnisse "/opt/cegcc/arm-cegcc/include/w32api/" und "/opt/mingw32ce/arm-mingw32ce/include/" kopieren.

Ecore

Um *Ecore* zu erstellen muss zu erst eine Änderung im "winnt.h"-Header vorgenommen werden. Dieser liegt im *include*-Verzeichniss des *mingw32ce*-Kompilers.

```
#define PROCESS_SET_QUOTA          0x0100
#define PROCESS_SET_INFORMATION    0x0200
#define PROCESS_QUERY_INFORMATION  0x0400
+#define PROCESS_SUSPEND_RESUME    0x0800
#define PROCESS_ALL_ACCESS          (STANDARD_RIGHTS_REQUIRED|SYNCHRONIZE|0xfff)

#define THREAD_TERMINATE           0x0001
```

Der mit "+" gekennzeichnete Eintrag "PROCESS_SUSPEND_RESUME" muss in die Datei "winnt.h" eingefügt werden.

Nachdem dieser Schritt ausgeführt wurde kann nun auch *Ecore* kompiliert werden. Dazu wird auch hier wieder das "autogen.sh" Skript ausgeführt.

```
./autogen.sh --prefix=$WINCE_PATH --host=arm-mingw32ce --disable-pthread
```

Nachdem dies erfolgreich ausgeführt wurde können nun auch die gleichen zwei Schritte wie bei den vorhergegangenen Programmen ausgeführt werden.

Edje

Auch hier gilt wieder, Dateien herunterladen.

```
svn co http://svn.enlightenment.org/svn/e/trunk/embryo
```

Nachdem die Dateien heruntergeladen wurden, muss auch hier wieder das Skript aufgerufen werden.

```
./autogen.sh --prefix=$WINCE_PATH --host=arm-mingw32ce
```

Die letzten beiden Schritte sind auch hier wieder kompilieren und installieren.

```
make ; make install
```

Elementary

Zuerst müssen auch hier die benötigten Daten heruntergeladen werden.

```
svn co http://svn.enlightenment.org/svn/e/trunk/TMP/st/elementary
```

Nun muss auch wieder das “autogen.sh” Skript heruntergeladen werden.

```
./autogen.sh --prefix=$WINCE_PATH --host=arm-mingw32ce --with-edje-cc=$WINCE_PATH/bi
```

Um zwei Fehlern vorzubeugen, welche beim Erstellen der Test-Files von *elementary* auftreten, muss man im in der Datei “Makefile.am” im Ordner “src/bin/” alle Vorkommnisse von “test_fileselector.c” entfernen und folgende Zeilen auskommentieren.

```
bin_PROGRAMS = elementary_test
if BUILD_QUICKLAUNCH
bin_PROGRAMS += elementary_quicklaunch elementary_run elementary_testql
endif
```

Nun kann das Programm auf gewohnte Art und Weise erstellt und installiert werden.

```
make ; make install
```

Weitere Schritte

Im Anschluss an das Erstellen dieser Programme muss nun noch ein Skript in WINCE_PATH angelegt und dessen Zugriffsrechte abgeändert werden.

```
touch efl_zip.sh
chmod 774 efl_zip.sh
```

In dieses Skript wird nun der Code eingefügt, welcher unter Anhang 4 zu finden ist. Bei Ausführung dieses Skripts werden die vorhandenen *DLL*'s nocheinmal komprimiert und alles in einen Ordner mit dem Namen "efl" kopiert. Im Anschluss wird der ganze Ordner noch in einem *Zip-Archiv* komprimiert. Möchte man nun noch eigene Anwendungen hinzufügen, so muss man diese nur in diesen "efl" Ordner hinzufügen und erneut komprimieren. Nun kann dieses Archiv auf das Mobile Gerät kopiert und entpackt werden.

Anhang 2

Archive für *Eet*:

- zlib-1.2.3-bin.tar.bz2:
<http://sourceforge.net/projects/cegccc/files/ported%20packages/zlib-1.2.3/zlib-1.2.3-bin.tar.bz2/download>
- zlib-1.2.3-dev.tar.bz2:
<http://sourceforge.net/projects/cegccc/files/ported%20packages/zlib-1.2.3/zlib-1.2.3-dev.tar.bz2/download>
- libjpeg-6b-bin.tar.bz2:
<http://sourceforge.net/projects/cegccc/files/ported%20packages/libjpeg-6b/libjpeg-6b-bin.tar.bz2/download>
- libjpeg-6b-dev.tar.bz2:
<http://sourceforge.net/projects/cegccc/files/ported%20packages/libjpeg-6b/libjpeg-6b-dev.tar.bz2/download>

Anhang 3

Archive für *Evas*:

- freetype-2.3.7-bin.tar.bz2:
<http://sourceforge.net/projects/cegcc/files/ported%20packages/freetype-2.3.7/freetype-2.3.7-bin.tar.bz2/download>
- freetype-2.3.7-dev.tar.bz2:
<http://sourceforge.net/projects/cegcc/files/ported%20packages/freetype-2.3.7/freetype-2.3.7-dev.tar.bz2/download>
- libpng-1.2.33-bin.tar.bz2:
<http://sourceforge.net/projects/cegcc/files/ported%20packages/libpng-1.2.33/libpng-1.2.33-bin.tar.bz2/download>
- libpng-1.2.33-dev.tar.bz2:
<http://sourceforge.net/projects/cegcc/files/ported%20packages/libpng-1.2.33/libpng-1.2.33-dev.tar.bz2/download>

Anhang 4

efl_zip.sh:

```
#!/bin/sh

rm -rf efl/
rm -f efl.zip

mkdir -p efl/eina/mp
mkdir -p efl/evas/modules/engines/buffer/mingw32ce-arm/
mkdir -p efl/evas/modules/engines/software_16/mingw32ce-arm/
mkdir -p efl/evas/modules/engines/software_16_wince/mingw32ce-arm/
mkdir -p efl/evas/modules/engines/software_generic/mingw32ce-arm/
mkdir -p efl/evas/modules/loaders/eet/mingw32ce-arm/
mkdir -p efl/evas/modules/loaders/jpeg/mingw32ce-arm/
mkdir -p efl/evas/modules/loaders/pmaps/mingw32ce-arm/
mkdir -p efl/evas/modules/loaders/png/mingw32ce-arm/
mkdir -p efl/evas/modules/loaders/xpm/mingw32ce-arm/
mkdir -p efl/evas/modules/savers/eet/mingw32ce-arm/
mkdir -p efl/evas/modules/savers/png/mingw32ce-arm/

cp bin/eet.exe efl/
cp bin/libdl-0.dll efl/
cp bin/libevil-0.dll efl/
cp bin/libeina-0.dll efl/
cp bin/libeet-1.dll efl/
```

```
cp bin/libevas-0.dll efl/  
cp bin/libecore-0.dll efl/  
cp bin/libecore_evas-0.dll efl/  
cp bin/libecore_job-0.dll efl/  
cp bin/libecore_wince-0.dll efl/  
cp bin/libembryo-0.dll efl/  
cp bin/libedje-0.dll efl/  
  
arm-mingw32ce-strip efl/libdl-0.dll  
arm-mingw32ce-strip efl/libevil-0.dll  
arm-mingw32ce-strip efl/libeina-0.dll  
arm-mingw32ce-strip efl/libeet-1.dll  
arm-mingw32ce-strip efl/libevas-0.dll  
arm-mingw32ce-strip efl/libecore-0.dll  
arm-mingw32ce-strip efl/libecore_evas-0.dll  
arm-mingw32ce-strip efl/libecore_job-0.dll  
arm-mingw32ce-strip efl/libecore_wince-0.dll  
arm-mingw32ce-strip efl/libembryo-0.dll  
arm-mingw32ce-strip efl/libedje-0.dll  
  
cp lib/eina/mp/eina_chained_mempool.dll efl/eina/mp  
cp lib/eina/mp/eina_fixed_bitmap.dll efl/eina/mp  
cp lib/eina/mp/eina_pass_through.dll efl/eina/mp  
  
arm-mingw32ce-strip efl/eina/mp/eina_chained_mempool.dll  
arm-mingw32ce-strip efl/eina/mp/eina_fixed_bitmap.dll  
arm-mingw32ce-strip efl/eina/mp/eina_pass_through.dll  
  
cp lib/evas/modules/engines/buffer/mingw32ce-arm/module.dll \  
efl/evas/modules/engines/buffer/mingw32ce-arm/engine_buffer.dll  
  
cp lib/evas/modules/engines/software_16/mingw32ce-arm/module.dll \  
efl/evas/modules/engines/software_16/mingw32ce-arm/  
engine_software_16.dll  
  
cp lib/evas/modules/engines/software_16_wince/mingw32ce-arm/module.dll \  
efl/evas/modules/engines/software_16_wince/mingw32ce-arm/  
engine_software_16_wince.dll  
  
cp lib/evas/modules/engines/software_generic/mingw32ce-arm/module.dll \  
efl/evas/modules/engines/software_generic/mingw32ce-arm/\  
engine_software_generic.dll
```

```
engine_software_generic.dll

cp lib/evas/modules/loaders/eet/mingw32ce-arm/module.dll \
efl/evas/modules/loaders/eet/mingw32ce-arm/loader_eet.dll

cp lib/evas/modules/loaders/jpeg/mingw32ce-arm/module.dll \
efl/evas/modules/loaders/jpeg/mingw32ce-arm/loader_jpeg.dll

cp lib/evas/modules/loaders/pmaps/mingw32ce-arm/module.dll \
efl/evas/modules/loaders/pmaps/mingw32ce-arm/loader_pmaps.dll

cp lib/evas/modules/loaders/png/mingw32ce-arm/module.dll \
efl/evas/modules/loaders/png/mingw32ce-arm/loader_png.dll

cp lib/evas/modules/loaders/xpm/mingw32ce-arm/module.dll \
efl/evas/modules/loaders/xpm/mingw32ce-arm/loader_xpm.dll

cp lib/evas/modules/savers/eet/mingw32ce-arm/module.dll \
efl/evas/modules/savers/eet/mingw32ce-arm/saver_eet.dll

cp lib/evas/modules/savers/png/mingw32ce-arm/module.dll \
efl/evas/modules/savers/png/mingw32ce-arm/saver_png.dll

arm-mingw32ce-strip efl/evas/modules/engines/buffer/\
mingw32ce-arm/engine_buffer.dll

arm-mingw32ce-strip efl/evas/modules/engines/software_16/\
mingw32ce-arm/engine_software_16.dll

arm-mingw32ce-strip efl/evas/modules/engines/\
software_16_wince/mingw32ce-arm/engine_software_16_wince.dll

arm-mingw32ce-strip efl/evas/modules/engines/\
software_generic/mingw32ce-arm/engine_software_generic.dll

arm-mingw32ce-strip efl/evas/modules/loaders/eet/mingw32ce-arm/loader_eet.dll
arm-mingw32ce-strip efl/evas/modules/loaders/jpeg/mingw32ce-arm/loader_jpeg.dll
arm-mingw32ce-strip efl/evas/modules/loaders/pmaps/mingw32ce-arm/loader_pmaps.dll
arm-mingw32ce-strip efl/evas/modules/loaders/png/mingw32ce-arm/loader_png.dll
arm-mingw32ce-strip efl/evas/modules/loaders/xpm/mingw32ce-arm/loader_xpm.dll

arm-mingw32ce-strip efl/evas/modules/savers/eet/mingw32ce-arm/saver_eet.dll
arm-mingw32ce-strip efl/evas/modules/savers/png/mingw32ce-arm/saver_png.dll
```

```
cp freetype-2.3.7-bin/bin/libfreetype-6.dll efl/  
cp libjpeg-6b-bin/bin/jpeg62.dll efl/  
cp libpng-1.2.33-bin/bin/libpng12-0.dll efl/  
cp libpng-1.2.33-bin/bin/libpng-3.dll efl/  
cp zlib-1.2.3-bin/bin/zlib1.dll efl/  
  
zip -r -9 efl.zip efl/
```

Literatur

- [Android] *Android*. – URL <http://www.android.com/>. – [Online; letzter Aufruf 25.01.2010]
- [CeGCC] *CeGCC*. – URL <http://cegcc.sourceforge.net/>. – [Online; letzter Aufruf 20.11.2009]
- [eFl] *Enlightenment*. – URL <http://www.enlightenment.org/>. – [Online; letzter Aufruf 20.11.2009]
- [Latitude] *Google Latitude*. – URL http://www.google.com/intl/en_us/latitude/intro.html. – [Online; letzter Aufruf 11.02.2010]
- [iPhoneOS] *iPhone OS*. – URL <http://www.apple.com/de/iphone/>. – [Online; letzter Aufruf 03.02.2010]
- [IRC] *IRC-Protokoll*. – URL <http://www.irc.org/>. – [Online; letzter Aufruf 27.01.2010]
- [IRCD] *IRCD-Hybrid*. – URL <http://www.ircd-hybrid.org/>. – [Online; letzter Aufruf 11.02.2010]
- [OpenSSL] *libCrypto*. – URL <http://www.openssl.org/>. – [Online; letzter Aufruf 25.01.2010]
- [libircclient] *libircclient*. – URL <http://libircclient.sourceforge.net/>. – [Online; letzter Aufruf 25.01.2010]
- [PNG] *libpng*. – URL <http://www.libpng.org/>. – [Online; letzter Aufruf 11.02.2010]
- [qrencode] *libqrencode*. – URL <http://megaui.net/fukuchi/works/qrencode/index.en.html>. – [Online; letzter Aufruf 11.02.2010]
- [OSM] *OpenStreetMap*. – URL <http://www.openstreetmap.de/>. – [Online; letzter Aufruf 13.02.2010]
- [PalmOS] *PalmOS*. – URL <http://www.palm.com/>. – [Online; letzter Aufruf 25.01.2010]
- [POSIX] *Portable Operating System Interface for Unix Layer*. – URL <http://standards.ieee.org/regauth/posix/>. – [Online; letzter Aufruf 11.02.2010]
- [SymbianOS] *Symbian OS*. – [Online; letzter Aufruf 03.02.2010]
- [WebOS] *WebOS*. – URL <http://palmwebos.org/>. – [Online; letzter Aufruf 25.01.2010]
- [Windows] *Windows Mobile*. – URL <http://www.microsoft.com/windowsmobile/de-de/default.aspx>. – [Online; letzter Aufruf 25.01.2010]

[Wireshark] *Wireshark*. – URL <http://www.wireshark.org/>. – [Online; letzter Aufruf 27.01.2010]

[xchat] *X-Chat*. – URL <http://xchat.org/>. – [Online; letzter Aufruf 11.02.2010]

[Quercia und Capra 2009] QUERCIA, Daniele ; CAPRA, Licia: *FriendSensing: Recommending Friends Using Mobile Phones*. 2009. – URL web.mit.edu/quercia/www/publications/friendSensing_short.pdf. – [Online; letzter Aufruf 27.01.2010]

[Rechert und Welke 2009] RECHERT, Klaus ; WELKE, Konstantin: *Spontaneous Privacy-Aware Location Sharing*. 2009