B A C H E L O R A R B E I T

Mobiler Friend Finder

Patrick Hornecker

betreut durch

Klaus Rechert

an der

Technischen Fakultät
der Albert-Ludwigs-Universität Freiburg

Inhaltsverzeichnis

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 sogennanten Smartphones, 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 Smartphones ergibt sich somit eine immense Menge an möglichen Anwendungsgebieten.

Auch positionsabhängige Dienste verbreiten sich, dank *GPS*-Funktionen der Smartphones, immer weiter. So ist es mit bestimmten Programmen zum Beispiel möglich eine zurückgelegte Strecke zu speichern oder den nächsten Supermarkt in der Nähe anzuzeigen. Für die Anbieter solcher Dienste stellen diese ein rentables Geschäftsmodell dar. So können sie anhand der Orte, an denen sich der Nutzer befindet, gezielt Werbung platzieren. Wenn Positionsdaten über die Zeit gespeichert werden, so kann ein Anbieter in der Lage sein ein Bewegungsprofil des Benutzers zu erstellen.

Somit ist es besonders im Bereich von Positionsdaten ein wichtiger Aspekt, Daten zu verschlüsseln. Hierzu sollten Algorithmen genutzt werden, welche das Verschlüsseln von Daten ohne zuviel Berechnungsaufwand durchführen, um den Akku des Gerätes zu schonen. Um Daten zu ver- und entschlüsseln werden Schlüssel benötigt, die verteilt werden müssen. Beim Verteilen dieser Schlüssel muss allerdings darauf geachtet werden, dass diese nur Teilnehmer erhalten für welche sie auch bestimmt sind, da der Datenaustausch ansonsten abgehört werden kann.
Auch im Hinblick auf das Speichern und Sammeln von Daten der Nutzer kann ein anderer Ansatz erdacht werden. So würde durch eine dezentrale Infrastruktur kein zentraler Knotenpunkt existieren über den der gesamte Datenverkehr läuft. Somit wäre man von einem Betreiber oder einer Institution unabhängiger und es wäre nicht mehr so leicht Informationen über einen Benutzer zusammenzutragen.

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 privacy* Software auf mobilen Geräten aufgezeigt. *Location privacy* wird von Beresford und Stanjano durch "...the ability to prevent other parties from learning one's current or past position" [Beresford und Stajano, 2003] definiert.
Des weiteren werden die Anforderungen, die an ein Programm dieser Art gestellt werden, analysiert.

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 rund um die eigene Position 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 von Anwendungen, die die eigene Position für Freunde sichtbar macht.
So bietet *Google* beispielsweise den Dienst *Google Latitude* [Latitude] an. Bei diesem Programm ist es möglich die Position von Freunden, die diesen Dienst auch 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 wurde in einer Befragung von Versuchspersonen herausgefunden, dass diese keine Bedenken im Bezug auf Positionsdaten und Privatsphäre haben [Kaasinen, 2003]. Da es aber immer mehr solcher Dienste gibt, sollte man gerade zum Schutz dieser Benutzer, Anwendungen entwickeln welche Wert auf die Wahrung *location privacy* legen.
Um die Positionsdaten auf optimale Art und Weiße zu schützen, sollten diese nur in einem verschlüsselten Format versendet werden. Positionsdaten stellen für den Nutzer sensible Daten dar, da sie viel über seine Gewohnheiten und täglichen Ablauf verraten können. Wenn man diese Daten unverschlüsselt versendet so begibt man sich in die Gefahr das regelmäßige Aufenthaltsorte erkannt werden oder man sogar ständig aufspürbar ist, was eine erhebliche Verletzung der *location privacy* darstellen würde.
Mit der Thematik von verschlüsselter Datenübertragung auf mobilen Geräten befasst sich eine Arbeit mit dem Titel *Spontaneous Privacy-Aware Location Sharing* [Welke und Rechert, 2009]. Hierfür wurde ein Dienst für mobile Geräte implementiert welcher Daten verschlüsselt an mehrere Nutzer sendet. Es wurde ein möglichst einfaches und verlässliches Protokoll entworfen, mit dem Ziel die Berechnungszeiten niedrig zu halten.

Ein anderes Problem stellt die Nutzung von zentral organisierten Netzen dar. Hier ist es möglich, ohne das der Benutzer davon Kentniss hat, auf alle über diesen Knoten versandten Daten zuzugreifen. Dies könnte durch die Nutzung eines dezentralen, verteilten Systems eingeschränkt werden. Somit würde nicht nur keine Kontrollinstanz existieren, welche Einsicht in die Daten der

Nutzer hat während die Nutzer selbst immer nur Zugang zu den für sie bestimmten Daten besitzen, sondern man könnte die Daten auf mehrere Knoten verteilen. Diese Tatsache würde das gezielte Sammeln von Daten erschweren und somit der oben genannten Defintion von *location privacy* entgegenkommen.

2.2 Anwendungsmöglichkeiten

Die Verbreitung von solchen mobilen Geräten hat in den letzten Jahren beständig zugenommen und alle neueren Modelle besitzen ein *GPS*-Modul. Des weiteren sind alle modernen Geräte in der Lage, Daten sowohl über den *3G*-Standart sowie per *WLAN* zu übertragen. Wenn nun eine Gruppe, welche mit entsprechenden mobilen Geräten ausgestattet ist, die Positionen austauschen möchte, so wird zuallererst ein Protokoll benötigt welches die Daten versenden kann. Dieses Protokoll sollte sowohl verlässlich als auch für langsame Netze konzipiert sein. Im nächsten Schritt soll diese Verbindung nun nur verschlüsselte Daten übertragen. Hierzu wird also ein geeigneter Algorithmus benötigt, welcher die Daten verschlüsselt. Dieser sollte möglichst wenig Berechnungsaufwand eine möglichst gute Verschlüsselung bieten.
Will man nun Daten verschlüsseln so benötigen sowohl Sender als auch Empfänger dieser Daten die gleichen Schlüssel. Es wird also auch eine Möglichkeit gesucht, mit deren Hilfe man Schlüssel auf eine sichere Art und Weiße übertragen kann. Sind diese Anforderungen erfüllt, so können die Mitglieder dieser Gruppe nun ihre Positionsdaten austauschen, ohne diese Möglichweise Personen mitzuteilen für die diese Daten nicht bestimmt sind.
Mit diesem gegebenen Protokoll und den Verschlüsselungsverfahren ist es nun ohne weiteres möglich auch Textnachrichten zwischen zwei Mitgliedern einer Gruppe auszutauschen.

2.3 Ziele

Die Anwendung eines solchen Programmes soll dem Nutzer möglichst einfach gemacht werden und trotzdem die gesetzten Ziele, wie die Verschlüsselung der Daten und Nutzung einer dezentralen Infrastruktur, erfüllen.

Es soll also sicher gestellt sein, dass es auch Benutzern ohne Fachkentniss möglich ist, die benötigten Schlüssel untereinander auszutauschen und somit festzulegen wer alles diese Positionsdaten einsehen darf. Des weiteren muss beim Schlüsselaustausch gewährleistet sein, dass der Schlüssel auf eine sichere Art und Weiße übertragen wird, ohne das andere diesen mitlesen 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 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.

2.4 Anforderungen

Anhanden der Anwendungsmöglichkeiten lassen sich die Anforderungen, die an die Software gestellt werden, ableiten. So sollte es möglich sein die Standorte anderer Nutzer anzuzeigen. Damit für andere Anwender die eigene Position sichtbar ist, muss diese in einem gängigen Format versendet werden. Hierfür ist das Format *Latitude/Longtitude* bestens geignet, da es sich um das am meisten verbreitetste Format handelt und es möglich ist jeden Punkt auf der Erdeerfläche ob in diesem Geokoordinatensystem darzustellen. Um zu verhindern dass die Daten von jedem gelesen werden können müssen diese Positionsdaten in verschlüsselter Form versendet werden.

Um die Position anderer Teilnehmer zu visualisieren sollte das Programm in der Lage sein, die eigehenden 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 darstelbar ist und man einfach auf den neusten Stand bringen kann. Es werden nur Benutzer innerhalb eines Radiuses angezeigt, da die Positionen von Teilnehmern, die sich in größerer Entfernung befinden, schwerer zu erreichen sind.

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

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. Es ist allerdings darauf zu achten, dass dieser Schlüssel nicht während der übertragung

abgefangen werden kann, da die Kommunikation ansonsten nichtmehr sicher wäre.

Auf dem Markt sind aktuell viele unterschiedliche Betriebsysteme für Smartphones vertreten. Um möglichst viele Benutzer zu erreichen sollte das Programm der Lage sein auf möglichst vielen dieser Plattformen zu arbeiten. Im Zuge dieser Plattformunabhängigkeit sollte die Software in einer Sprache implementiert werden, die von möglichst vielen Betriebssystemen der mobilen Geräte unterstützt wird.

Die Struktur des Programmes sollte möglichst modular gehalten werden, damit es auch in späteren Phasen den Entwicklern leicht fällt bestimmte Programmteile auszutauschen. Somit soll gewährleistet werden das bestimmte Funktionen auch zu einem späteren Zeitpunkt ausgetauscht werden können. So wäre es zum Beispiel denkbar verschiedene Algorithmen zur Verschlüsselung oder ein anderes Protokoll zum Versenden der Daten zusätzlich zu implementieren oder die bisher genutzten Algrorithmen auszutauschen.

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 des 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. Des weiteren sind symmetrische Verfahren nicht so berrechnungsintensiv wie asymmetrische, was einen wichtigen Punkt auf mobilen Geräten darstellt.

Der Austausch der Schlüssel könnte theoretisch auf mehreren Wegen geschehen. So könnte man sie per *Bluetooth* übertragen oder einen 2D-Barcode [qrcode] aus der Zeichenkette erstellen, fotographieren und wieder in eine Zeichenkette umwandeln. Da *Bluetooth* ein unsicheres Medium darstellt [Shaked und Wool, 2005], der Sitzungs PIN kann per Daten-Phishing wiederhergestellt werden, werden die Schlüssel per Barcode verteilt. Es findet somit keine Datenübertragung durch Kommunikation zwischen den Geräten statt, womit der Schlüssel auf diesem Weg nicht abgefangen werden kann.

Für die Kommunikation zwischen den einzelnen Teilnehmern ist ein dezentrales Protokoll von Nöten, 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.
Ein Vorhandenes Protokoll, welches ein aktives Netzwerk bereitstellt das auch rege genutzt wird, ist das *IRC*-Protokoll [IRC]. Es stehen bereits mehrere Server zur Verfügung und jeder Nutzer kann einen eigenen Server bereitstellen. 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.
Auf dieses Verfahren soll nun das Protokoll zum Versenden von Textnachrichten sowie Positionsdaten aufsetzten. Die Daten werden in beiden Fällen verschlüsselt über einen *IRC*-Channel gesendet und dort ausgegeben. Beim Versenden der Positionen muss zwischen Sender und Empfänger unterschieden werden. Dies geschieht, indem dem User ein entsprechender Suffix angehängt wird. Somit kann zwischen diesen beiden Diensten unterschieden werden. Beim Kommunizieren mit Textnachrichten muss der Benutzer im Vorfeld festlegen mit welchem anderen Teilnehmer er Nachrichten austauschen möchte. Daraufhin werden den zwei Anwendern nur die jeweiligen Textnachrichten des anderen aus dem *Channel* angezeigt.

Durch die Wahl dieser Lösung ist sowohl die Berechnungszeiten durch ein symmetrisches Verfahren niedrig gehalten, der Verwaltungsaufwand für die Schlüssel gering 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 die *IRC*-Server nur in verschlüsselter Form. Des weiteren kann jeder beliebige *IRC*-Server gewählt werden. Somit sind die Eingangs erwähnten Punkte, Verschlüsselung und dezentralles Protokoll, hiermit abgedeckt und die Privatsphäre ist für den Anwender, im Unterschied zu den meisten anderen Anwendungen dieser Art, gesichert.

3 Technische Grundlagen

Die Wahl der Plattform hängt von zwei verschiedenen Faktoren ab. Zum einen stellt sich die Frage, ob die Handymodelle die benötigte Hardware, wie zum Beispiel *GPS* oder eine Kamera besitzen, zum anderen ob Schnittstellen vorhanden sind um das Programm für das System zu entwickeln.

Die Problematik der Plattformwahl aufgrund von vorhandener oder nicht vorhandener Hardware ist nicht allzu groß. Die meisten aktuellen Geräte haben mittlerweile 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.

Die Wahl der Plattform aufgrund des Betriebssystemes gestalltet sich schon schwerer. Bei geeigneter Auswahl ist es möglich die Software auf mehrere Betriebssysteme für Smartphones 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. Es wäre also ein *Layer* interessant, welchen man mit den immer gleichen Programmbibliotheken nutzen kann, unabhängig was diesem *Layer* für ein System zu Grunde liegt. Dieser *Layer* soll also eine Schnittstelle zwischen Anwendungen und Betriebssytem repräsentieren. Ein *Layer* welcher genau diese Anforderungen erfüllt 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. Diese haben den Vorteil, dass sie aktiv weiterentwickelt werden und auch ständig neue Bibliotheken hinzukommen. 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.

3.1 Betriebsysteme für mobile Geräte

Wie schon erwähnt ist die Wahl einer geeigneten Plattform von nicht unerheblicher Wichtigkeit, da hier schon indirekt eine Vorauswahl an Nutzbaren Bibliotheken und Programmiersprachen getroffen wird. 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 mit *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 eine 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 desen 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* laut Hersteller stabil auf den Geräten laufen. 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 *WebOS* einen einen *POSIX Layers*. 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* oder einer anderen Abstraktion dieser Art. 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 eine 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 weitern besitzt *Symian 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 zu portieren.
iPhoneOS wurde aufgrund seiner mangelnden *Multitasking*-Unterstützung ausgeschlossen. Diese ist für den geplanten Dienst wichtig, da hier Prozesse im Hintergund 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

Anhand 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, bassierend 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 auch *Windows Mobile* Bibliotheken zum Einstatz kommen.

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.

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 Bibliothek, welche das serialisieren von mehreren Programmteilen ermöglicht und für den Betrieb auf mobilen Geräten optimiert wurde. *Edje* ist eine komplexe 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

Abbildung 1: Beispiele verschiedener *Elementary* Icons

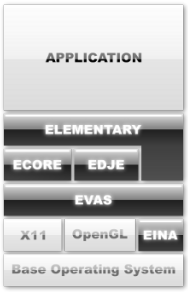

Abbildung 2: Aufbau von *Enlightenment*

Datentypen und Tools bereit.

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

4 Friend Finder

Die Eingangs beschriebene Software trägt den Namen *Friend Finder* und wurde im Rahmen dieser Arbeit mit fast allen aufgezählten Funktionen realisiert. 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.

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 gesammte Programmcode der Benutzeroberfläche wurde in einer Datei zusammengefast (*gui.c*). Diese Tatsache vereinfacht das Erhalten der Modularität, da nur diese Datei durch eine andere ersetzt werden muss um einen anderen Typ von Oberfläche zu nutzen.
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

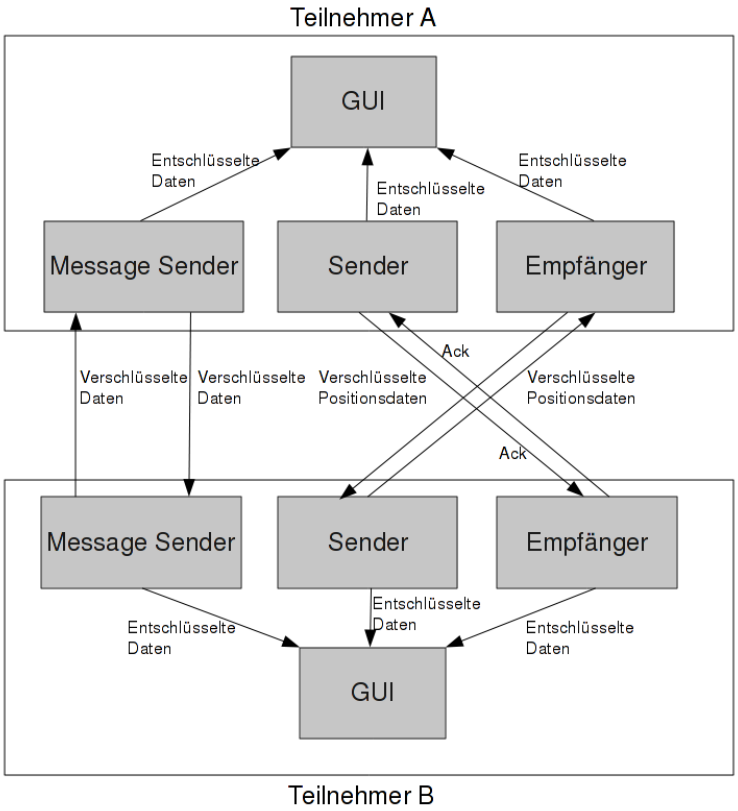

Abbildung 3: *Friend Finder* Nachrichtenaustausch

dessen Stabilität. Ein weiterer Vorteil ist, dass man Daten die an mehrere Benutzer gesendet werden sollen, nur einmal an einen *Channel* senden muss und jeder Benutzer in diesem *Channel* diese Daten empfangen kann.

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* [Schneier, 1994] wurde seitens der *OpenSSL*[OpenSSL] Bibliothek als *Blockcipher* implementiert. *Blowfish* wurde Aufgrund der schnellen verschlüsselungsrate sowie einfachen Implementierung gewählt und ist ein symmetrisches Verschlüsselungsverfahren. Das bedeutet, das 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.

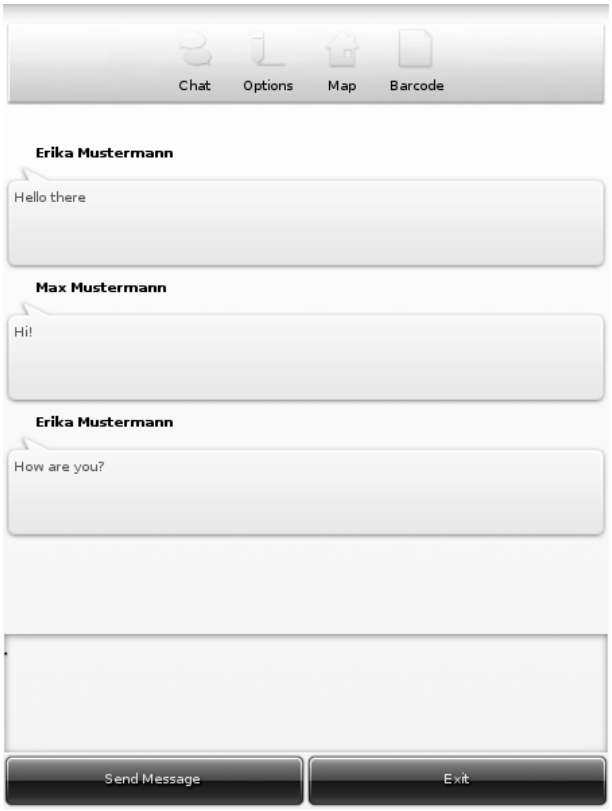

Abbildung 4: Versenden von Chatnachrichten

Um Nachrichten zu versenden wurde für dieses Projekt die *IRC-Client Bibliothek*[libircclient] verwendet. Diese Bibliothek 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 Breitengrade 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. Es werden für das Versenden einer Position insgesamt vier 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 wird ebenfalls verschlüsselt versandt. Kommt dieses *Acknowledgement* beim Sender an, so versendet dieser wieder ein *Latitude/Longtitude* 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, dass der Empfänger die Daten dem jeweiligen Absender zuordnen kann und diese auch wieder korrekt zusammensetzt. Um die einzelnen Fragmente wieder zuzordnen, werden an diese vor dem versenden Terminierungszeichen angefügt, welche zum Beispiel eindeutig als ersten Teil einer *Longtitude*-Koordinate identifizieren. Wenn diese wieder zugeordnet wurden und sowohl *Latitude* als auch *Longtitude* 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* angewandt.

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 im Anschluss 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 generiert. Die untere Abbildung zeigt einen solchen erstellten Barcode, wie er von *Friend Finder* ausgegeben wird.

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

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

Im folgenden Teil wird der erzeugte Datenverkehr von *Friend Finder* analysiert. Ein Hauptaugenmerkt wird hierbei vor allem auf die Packetgröße, 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 eine normalen *IRC-Clients*, hier *X-Chat* [xchat], ergibt 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* beitritt, so sendet dieser drei Pakete an den Server. Diese beinhalten den Namen des *Channels* dem beigetreten werden soll, eine Anfrage nach einer Liste 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 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, dass 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 besteht somit immer Datenverkehr zwischen Sender und *IRC*-Server und es kann schnell festgestellt werden ob eine Verbindung noch aktiv 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 Worldïn 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 hineingepakt 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 gepakt. 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-Fragmeint 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.
In der folgenden Abbildung ist an der X-Achse die Anzahl der Teilnehmer und an der Y-Achse die Anzahl der zu versendenden Dateneineheiten abgetragen. Hier ist deutlich zu sehen, dass bei steigender Teilnehmer Anzahl eine Lösung mit einer offenen Verbindung pro Benutzer soviel Datenpakete wie Nutzer versenden muss. Bei der Lösung mit einem zentralen *Channel* müssen die Daten immer nur einmal versandt werden, unabhängig von der Nutzeranzahl.

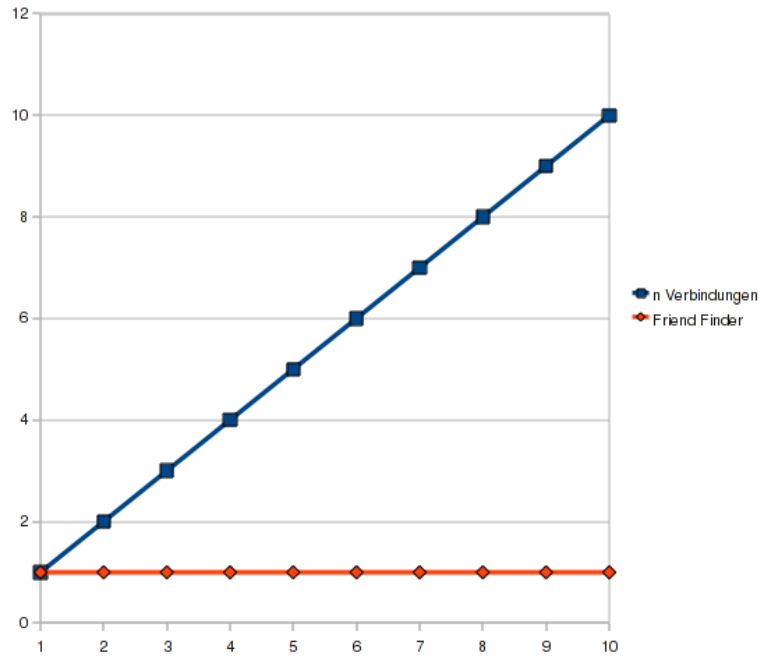

Abbildung 6: Vergleich von n-Verbindungen und *Friend Finder*

5 Fazit

Anhaden der Anforderungen und der implementierten Features von *Friend Finder* ist ersichtbar das es möglich ist einen sicheren Dienst, welcher die *location privacy* achtet, 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 diese 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 die Software in unterschiedlichen Sprachen und mit unterschiedlichen Bibliotheken implementieren zu müssen. Gerade diese und die Tatsache das die Betriebsysteme für mobile Geräte immer mehr die gleichen Schnittstellen nutzen, 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 Systemverzeichniss entpackt werden.

Bevor man mit dem nächsten Schritt fortfahren kann, müssen noch ein paar benötigte Packete 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 Packete aus dem *Subversion-Repository* der Entwickler herunterladen.
Nun muss man sich noch ein Verzeichniss 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 Kompiler 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$WINCE_PATH/win_icon-dev/lib
-L$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*, beziehungsweiße 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 geschen, die Datei mit den *Umgebungsvaribalen* 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 letzen 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 Zielverzeichniss 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 Packetinformationen 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 Ordnerebene 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 Verzichnisse "/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/bin/edje_cc
```

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

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]

[qrcode] *QR Code*. – URL http://www.denso-wave.com/qrcode/qrstandard-e.html. – [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.mspx. – [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]

[Beresford und Stajano 2003] BERESFORD, Alastair R. ; STAJANO, Frank: Location privacy in pervasive computing. In: *IEEE Pervasive Computing Magazine* (2003), S. 46–55

[Kaasinen 2003] KAASINEN, Eija: User needs for location-aware mobile services. In: *Personal and Ubiquitous Computing* Volume 7 (2003), Nr. 1, S. 70–79

[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]

[Schneier 1994] SCHNEIER, Bruce: Description of a New Variable-Length Key, 64-Bit Block Cipher (Blowfish). In: *Fast Software Encryption, Cambridge Security Workshop Proceedings* (1994), S. 191–204

[Shaked und Wool 2005] SHAKED, Yaniv ; WOOL, Avishai: Cracking the Bluetooth PIN. In: *Proceedings of the 3rd international conference on Mobile systems, applications, and services* (2005), S. 39–50

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