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 Entwicklung 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. Aktuellen Modellen, dieser sogennanten Smartphones, ist es zum Beispiel möglich sich mit einem *WLAN* zu verbinden, die eigene Position mittels *GPS* zu ermitteln oder per *3G*-Standard 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. Allerdings kann die so entstehende Datenflut auch anderweitig genutzt werden. So können anhand der Orte, an denen sich der Anwender befindet, gezielt Werbung platzieren werden, oder die Positionsdaten können über die Zeit gespeichert und Bewegungsprofile des Benutzers erstellt werden.
Anwendungen dieser Art können also einen nützlichen Dienst erbringen, allerdings können die entstehenden Daten auch missbraucht werden. Es ist also wichtig das die Positionsinformationen, die der Benutzer bereitstellt, als ein Teil seiner Privatsphäre angesehen werden.
Das Ziel dieser Arbeit ist die Implementation einer Anwendung für mobile Geräte, welcher es ermöglicht anderen Benutzern die eigene Position zu senden, sowie deren anzuzeigen. Hierbei soll für den Benutzer möglichst viel Transparenz geboten sein, so dass er immer in der Lage ist nachzuvollziehen was mit seinen Daten geschieht. Diese sollen allerdings trotzdem durch Verschlüsselung soweit geschützt werden, als das der Anwender bestimmen kann für wenn diese einsehbar sind.

2 Grundlagen

Location privacy wird von Duckham und Kulik durch

> ... a special type of information privacy which concerns the claim of individuals to determine for themselves when, how, and to what extent location information about them is communicated to others.

[2] definiert. Ein Anwender sollte also in der Lage sein den Zeitpunkt, wie und in welchem Umfang Positionsdaten über sie verbreitet werden, selbst festzulegen. Es stellt sich die Frage, wie der Benutzer in der Lage ist dieser drei Punkte zu kontrollieren. Dabei soll er den Zeitpunkt frei bestimmen könnnen, wann und auf welche Art und Weiße er die Daten versendet und wieviele andere Benutzer oder Institutionen darauf Zugriff haben. Im ersten Teil wird die aktuelle Entwicklung von *location privacy* Software auf mobilen Geräten aufgezeigt. Des weiteren werden die Anforderungen, die an ein Programm dieser Art gestellt werden, analysiert und mögliche Ziele einer solchen Software beschrieben.

2.1 Aktuelle Entwicklungen

Da so gut wie alle aktuellen Smartphones mit dem *Global Positioning System (GPS)* ausgestattet sind, so gibt es für die verschiedenen Betriebssysteme mittlerweile eine Reihe von Anwendungen die Funktionalitäten rund um die eigene Position bieten. So gibt es Anwendungen um sich Routen erstellen zu lassen, die eigene Position zu bestimmen oder um *Geocaching* [10] zu betreiben.
Zum Beispiel bietet *Google* den Dienst *Google Latitude* [1] 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.
Betrachtet man ein solches Programm unter den obigen Gesichtspunkten von Duckham und Kulik, so stellt man fest dass der Anwender hier nur den Zeitpunkt, zu dem die Positionsinformationen versendet werden, bestimmen kann. Nutzen Anwender das Programm, so werden die Daten an den Anbieter eines solchen Dienstes gesendet. Diesem Anbieter obliegen die Rechte über die Daten ab diesem Zeitpunkt. Die einzige Einsicht die der Benutzer in diesen Vorgang hat, ist die dass er befreundete Teilnehmer auf seinem Display und diese ihn auf ihrem Display sehen. Er hat also keinerlei Kontrolle darüber, was mit den Daten nach dem Absenden passiert, da diese an den Anbieter gesendet werden, wo sie dann durch ein interenes, ihm unbekanntes System an die anderen Nutzern weitergereicht werden. Wie dieses im genauen abläuft, weiß der Anwender nicht. Der Benutzer kann also weder festlegen, wie noch in welchem Umfang er die Daten versenden möchte, da er keinen Einblick in diesen Teil der Strukturen der Institution, die den

[1] http://www.google.com/intl/en_us/latitude/intro.html

Dienst zur Verfügung stellt, hat.
An genau diesem Punkt schließt die Arbeit *Spontaneous Privacy-Aware Location Sharing* [11] an. Hier werden die Daten über ein offenes und frei zugängliches System versendet. Da, dass zum Versenden genutzte, System offen zugänglich ist, müssen diese Daten zusätzlich verschlüsselt werden, da ansonsten jederman Zugang zu ihnen hätte.
Zum Verschleiern der Position nutzen Kido u.a. [4] Datensätze die falsche Positionsangaben beinhalten. Diese werden an einen Dienst gesandt, welcher auf all diese Datensätze antwortet, egal ob diese die richtigen oder die falschen Daten beinhalten. Nur der Client weiß, welche der empfangenen Daten auf der eigentlichen Position basieren. Mit dieser Lösung ist es zwar möglich Positionsdaten zu verschleiern, allerdings wird hiermit nicht das Problem gelöst, dass der Nutzer keine Kontrolle darüber hat, wie seine Daten versendet und genutzt werden.
Einen anderer Ansatz verfolgen Gruteser und Grundwald [3]. Sie versuchen mit Hilfe von *k-anonymity* [1] die Position zu verschleiern. Man versteht unter *k-anonymity*, dass in einer Menge von k Personen ein Teilnehmer nicht von den anderen $k-1$ Teilnehmern unterschieden werden kann. Gruteser und Grundwald haben hierfür einen *quadtree* [7] genutzt um bestimmte Bereiche zu erstellen. Diese Bereiche haben nur eine Vorraussetzung, sie müssen k Personen enthalten. Somit kann nicht festgestellt werden, welche Person eines Bereiches die Daten versandt hat. Allerdings kann dieses Vorgehen nicht für eine Anwendung mit der Funktionalität eines Dienstes, der gezielt Positionen einzelner andere Benutzer anzeigt, genutzt werden. Der Grund dafür ist, dass es mit diesem Ansatz nicht möglich ist Positionsdaten von einzelnen Personen zu versenden. Zusätzlich ist auch hier nicht gegeben, dass die Anwender Einsicht in die Verwendung und Verbreitung ihrer Daten erhalten.

2.2 Vorraussetzungen

Im Rahmen der Datenübertragung sind alle modernen Geräte in der Lage, Daten sowohl über den *3G*-Standard sowie per *WLAN* zu übertragen.
Wenn nun ein sicherer Austausch von Positionsdaten erfolgen soll, so müssen neben der Hardware, auch andere Rahmenbedingungen gegeben sein. Da man durch Positionsdaten die aktuelle Position erfahren oder Bewegungsprofile erstellen kann, sollten der Zugang zu diesen nur dann erlaubt sein wenn der Benutzer der sie versendet damit einverstanden ist. Die Daten müssen also soweit entfremdet oder abgeändert werden damit nur eine vom Nutzer bestimmte Gruppe diese wieder rekonstruieren kann. Somit ist gegeben, dass der Nutzer über das Ausmaß der Verbreitung seiner Daten die Kontrolle bewahren kann.
Der Anwender muss Daten zu jedem von ihm gewünschten Zeitpunkt zu versenden können. Er muss also in der Lage sein die dafür benötigten Parameter zu sofort zu erstellen und weiterzugeben. Somit kann er den Zeitpunkt, zu welchem er seine Daten versenden möchte, frei wählen.
Ist dies gegeben, werden die Daten im nächsten Schritt versendet. Auch hier muss ein Weg gefunden werden mit dem der Nutzer möglichst viel Kontrolle über seine Datensätze inne hat. Um diese Kontrolle zu wahren, müssen die Informationen mit einer möglichst transparenten und doch

verlässlichen Methode verschickt werden. Zum Weiteren Schutz der verschlüsselten Daten darf man zum Übertragen der Informationen nicht nur auf einen zentralen Knoten angewiesen sein, sondern nutzt im optimalen Fall ein ganzes Netzwerk von solchen Knotenpunkten. Bei diesem Netzwerk sind allerdings alle Knotenpunkt einsehbar, damit der Nutzer zu jedem Zeitpunkt weiß, was mit seinen Daten geschieht. Ist dies gegeben so ist der Nutzer in der Lage zu kontrollieren wie seine Daten versandt werden.

2.3 Ziele

Zusammenfassend kann also gesagt werden, dass eine Software mit den beschriebenen Eigenschaften die Sicherheit der Daten, in Bezug auf Zugänglichkeit, und das Vermeiden von Datenspeicherung zum Ziel hat. Anwender soll es aber trotz allem einfach gemacht werden dieses Programm zu nutzen. Die Inhalte der Anwendung müssen also soweit abstrahiert werden, als dass auch ein Benutzer ohne Fachkenntnis alle Funktionen der Anwendung nutzen kann, ohne dass die obigen Punkte ausser Kraft treten.
Zum Versenden der Daten muss eine offene Struktur genutzt werden, in welche jeder Einsicht hat und welche frei genutzt werden kann. Es muss gewährleistet sein das jeder Teilnehmer dieser sofort beitreten kann, ohne das für ihn Restriktionen, wie zum Beispiel ein Benutzeraccount, gelten. Diese Struktur sollte über ein Protokoll verfügen das verlässlich, stabil und auch für langsame Netzwerke optimiert ist. Eine reibungslose Kommunikation kann somit garantiert werden.
Um die Kommunikation zwischen verschiedenen Teilnehmern zu ermöglichen muss es möglich sein Chatnachrichten auszutauschen. Die Interaktion zwischen den Anwendern und somit der Nutzen des Dienstes kann hierdurch weiter gesteigert werden.
Durch die Nutzung einer, für alle, offene Struktur ist es von Nöten, dass die Daten verschlüsselt werden. Da bei Verschlüsselungen der Austausch von Schlüsseln voraussetzt wird, muss ein Verfahren genutzt werden welches den Austausch von zwei Schlüsseln auf einfache Weise ermöglicht. Da diese Software für mobile Geräte ausgelegt ist, müssen die benötigten Schlüssel spontan ausgetauscht werden können, da der Anwender die Nutzung eines solchen Dienstes nicht immer im Vorraus planen möchte und kann. Gleichzeitig muss garantiert sein dass die Schlüssel während des Austauschs nicht von unbefugten Personen abgefangen werden.
Ist ein solches Verfahren gegeben, so kann ein Algorithmus genutzt werden der sowohl sicher ist, als auch mit möglichst geringem Aufwand die Daten ver- und entschlüsselt.
Da Positionsdaten versendet werden, müssen diese visualisiert werden. Somit muss die Applikation in der Lage sein die Standorte anderer Nutzer anzuzeigen. Da aus Gründen der Nutzbarkeit die Positionen der Teilnehmer auf einer Karte dargestellt werden sollen, muss ein Format für die Karte genutzt werden, welches auf dem mobilen Gerät darstellbar und einfach auf den neusten Stand zu bringen ist.
Benutzer die sich in größeren Entfernungen befinden sind für Programme dieser Art nur begrenzt interessant, da sie mit zunehmender Entfernung immer schwerer zu erreichen sind. Deshalb werden nur Teilnehmer innerhalb eines bestimmten Radius angezeigt.
Ein weiterer Punkt ist die Plattformunabhängigkeit. Diese steht zwar nicht in Verbindung mit

der Privatsphäre der Benutzer, allerdings sollte ein solches Programm unter möglichst vielen Plattformumgebungen lauffähig sein. Zum Einen wird somit der Aufwand der Implementierung verringert, da die Software nicht mehrmals implementiert werden muss. Zum Anderen werden möglichst viele Benutzer erreicht und es kann auch Kommunikation unter Besitzern von unterschiedlichen Typen von Mobiltelefonen stattfinden.
Die Struktur der Software muss möglichst modular gehalten werden, damit es auch zu einem späteren Zeitpunkt leicht fällt Programmteile auszutauschen oder zu erweitern. So wäre es zum Beispiel denkbar verschiedene Algorithmen zur Verschlüsselung oder ein anderes Protokoll zum Versenden der Daten, zusätzlich zu implementieren.

2.4 Verfahren

Anhand der Anforderungen müssen nun geeignete Verfahren und Protokolle sowohl für Kommunikation als auch für Verschlüsselung gewählt werden. Da man mit mehreren Benutzern oder auch mehreren Benutzergruppen kommunizieren kann, können mehrere Schlüssel anfallen. Somit soll der Aufwand, für den Anwender, um diese Schlüssel zu speichern, zu löschen oder neu zuzuordnen möglichst gering gehalten werden. Symmetrische Verfahren nicht so berrechnungsintensiv wie asymmetrische [5], was einen wichtigen Punkt auf mobilen Geräten darstellt. Aufgrund dieser Tatsache wird eine symmetrische Verschlüsselung genutzt.
Da der Schlüsselaustausch spontan durchführbar sein soll, muss dieser zu jedem Zeitpunkt möglich sein ohne das dafür Vorbereitungen getroffen werden müssen. So könnte man die Schlüssel per *Bluetooth* übertragen, da eine solche Verbindung ohne Vorarbeit aufgebaut werden kann. Allerdings stellt *Bluetooth* ein unsicheres Medium dar [9], da der *Bluetooth-Sitzungs-PIN* per *Daten-Phishing* wiederhergestellt werden kann. Eine andere Möglichkeit, die ebenfalls keine Vorarbeit benötigt, ist das Erstellen eines 2D-Barcodes [2] aus einer Zeichenkette. Dieser kann fotographieren und wieder in eine Zeichenkette umwandelt werden. Da keine Kommunikation über einen unsicheren Kanal zwischen den Geräten stattfindet sind die Barcodes optimal zum Schlüsselaustausch geeignet, da der Schlüssel auf diesem Wege nicht abgefangen werden kann.
Aufgrund der Möglichkeit das jeder Nutzer beliebig beitreten und Daten in dieser Struktur austauschen kann, fiel die Wahl zum versenden der Daten auf das *IRC*-Protokoll. Des weiteren spricht für diese Entscheidung, dass das *IRC*-Protokoll [6] weit verbreitet ist und eine ausgedehnte Serverstruktur zu Grunde liegt. Auch die Stabilität und Verlässlichkeit des Protokolles ist gegeben. Da die *IRC*-Server in Netzwerken organisiert sind führt der Ausfall eines Servers nicht zur Beendigung der gesamten Kommuniktion. Innerhalb der *IRC*-Netzwerke werden verschiedene *Channels* bereitgestellt, an welche Nachrichten gesendet werden können. Ein weiterer Vorteil von *IRC* ist, wenn Daten an mehrere Benutzer gesendet werden sollen, diese nur einmal an einen *Channel* versandt werden müssen und jeder Benutzer in diesem *Channel* diese Daten empfangen kann. Des weiteren steht es jedem Benutzer frei, eigene *Channels* zu öffnen.
In der beschriebenen Software werden die Positionsdaten als Zeichenfolge an einen dieser *Channels* gesendet und können dort von beliebig vielen anderen Instanzen der Software ausgelesen

[2] QR Code http://www.denso-wave.com/qrcode/qrstandard-e.html

werden. Diese verarbeiten die Daten im Anschluss, so dass diese dann als Position auf einer Karte ausgegeben werden können. Beim Versenden der Textnachrichten ist die vorgehensweiße equivalent.

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.
Da die gegebenen Hardwareunterschiede minimal sind, wird eine Plattform aufgrund des Betriebssystemes ausgewählt. Bei geeigneter Wahl 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.
Ein weiterer Punkt der zu einer Entscheidung beiträgt ist die Frage ob andere Programme und Bibliotheken auf den jeweiligen Systemen ausführbar sind. Es wird also ein *Layer* benötigt, mit welchem immer die gleichen Programmbibliotheken genutzen werden können. Dabei sollte das zugrundeliegende System unabhängig von diesem *Layer* sein. Es soll also damit vom zugrundeliegenden Betriebssystem abstrahiert werden. Ein entsprechender *Layer* stellt der *Portable Operating System Interface for Unix Layer (POSIX Layer)* [3] dar. 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, ohne das die genutzten Bibliotheken ausgetauscht werden müssen.
Zusätzlich stellt sich auch die Frage der zu nutzenden Programmiersprache. Diese sollte von einer möglichst großen Mengen an Betriebsystemen unterstützt werden. Man müsste das, wenn man das Programm für ein neues Gerät portieren möchte, das Progamm somit nicht immer komplett neu implementieren.

3.1 Betriebsysteme für mobile Geräte

Durch die Wahl eines Betriebsystemes wird 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] POSIX http://standards.ieee.org/regauth/posix/

3.1.1 Windows Mobile

Das von Microsoft entwickelte *Windows Mobile* [4] in der aktuellen Version 6.5 verfügbar. Das gesamte Betriebssystem basiert auf der *Windows Win32 API* und lässt Ähnlichkeiten zu den Desktop-Varianten der Windows-Familie erkennen. Es existiert ein *Cross-Compiler* names *CeGCC* [5], mit welchem Programme die in *C*/*C++* geschrieben wurden für diese Plattform kompiliert werden können.

3.1.2 Android

Das von *Google* entwickelte *Android* [6] 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 Anzahl der nutzbaren Sprachen im Moment eben auf diese eine beschränkt. Des weiteren bietet *Google* ein *NDK* an, mit desen Hilfe es auch möglich ist Programme in *C* oder *C++* zu schreiben.

3.1.3 WebOS

WebOS [7] wurde von *Palm* als Nachfolger von *PalmOS* [8] entwickelt und ist momentan nur auf zwei Geräten zu finden: Auf dem *Palm Pre* und dem *Palm Pixi*.
Unter der Benutzeroberfläche von *WebOS* arbeitet ein *Linux-Kernel* in der Version 2.6. Somit ist es möglich, wenn man Zugriff auf diesen Kernel hat, *POSIX*-kompatible Software unter *WebOS* zu betreiben. Für dieses Betriebssystem existiert ein *SDK* für *HTML5*, *CSS* und *Java*. Ein weiteres, welches im März 2010 veröffentlicht wird, soll die *C* und *C++* Entwicklung ermöglichen.

3.1.4 iPhone OS

Bei *iPhoneOS* [9] handelt es sich um eine abgeänderte und angepasste Version von MacOS. Somit bassiert auch der Kernel von *iPhoneOS* sowohl auf teilen eines *BSD*- [10] sowie *Mach*-Kernels [11].

[4] Windows Mobile http://www.microsoft.com/windowsmobile/de-de/default.mspx
[5] CeGCC http://cegcc.sourceforge.net/
[6] Android http://www.android.com/
[7] WebOS http://palmwebos.org/
[8] PalmOS http://www.palm.com/
[9] iPhoneOS http://www.apple.com/de/iphone/
[10] BSD https://www.bsdwiki.de/
[11] Mach http://www.cs.cmu.edu/afs/cs/project/mach/public/www/mach.html

Dieses Betriebssystem wurde eigens für das iPhone entwickelt. Auch für dieses System existiert ein *SDK*, welches allerdings nur die Sprache *Objective-C* unterstützt. Der größte Kritikpunkt an diesem System ist das Fehlen von *Multitasking*-Unterstützung. 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 [12] 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* einen *POSIX Layer*.

3.1.6 Zielplattform

iPhoneOS wurde aufgrund seiner mangelnden *Multitasking*-Unterstützung ausgeschlossen. Diese ist für den geplanten Dienst wichtig, da bei diesem Prozesse im Hintergund notwendig sind und dies auf einem solchen System nicht realisierbar wäre. *Android* hat zwar eine *C* Unterstützung, allerdings gibt der Hersteller an das nur die mit dem *NDK* verfügbaren Bibliotheken zum momentanen Zeitpunkt stabil verfügbar sind. Dies schränkt die Entwicklung stark ein und ist für das zu entwickelnde Programm nicht geeignet.
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

Mit dem *CeGCC* 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

[12] SymbianOS http://www.symbian.org/

auf dem *GCC*. Mit diesem Tool können in einer *Linux* Umgebung die für *Windows Mobile* benötigten Bibliotheken und ausführbaren Dateien erstellt werden.
Der *CeGCC* kann in zwei unterschiedlichen Versionen genutzt werden. Zum einen bietet der *wince-mingw32ce* eine Menge an Tools mit welchen man native *Windows Mobile* Applikationen erstellen kann. Hierzu wird eine Portierung der *GNU* [13] Enwicklungswergzeuge aus dem *MinGW* [14] Projekt genutzt. Diese kompilieren und linken Code, um diesen unter *Windows Mobile* ausführbar zu machen. Die andere Möglichkeit stellt die Nutzung des *arm-cegcc* dar. Mit diesem kann *POSIX* kompatibler Programmcoder nach *Windows Mobile* portiert werden.

3.2.2 Enlightenment

Neben einem *Cross-Compiler* wird noch ein geeignetes Frontend benötigt, um das Programm auch für den Benutzer ansprechend darzustellen und 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 erhalten. Genutzt wird das freie, seit 1997 existierende, *Enlightenment* [15] 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.

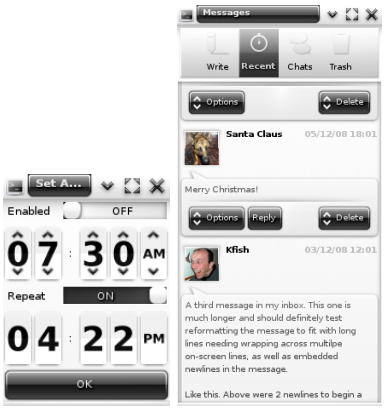

Abbildung 1: Beispiele verschiedener *Elementary* Icons

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.

[13] GNU http://www.gnu.org/
[14] MinGW http://mingw.org/
[15] Elementray http://www.enlightenment.org/

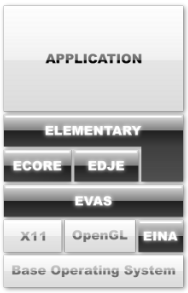

Abbildung 2: Aufbau von *Enlightenment*

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 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 mit deren Hilfe es möglich ist *Elementary* nach *Windows Mobile* zu portieren.

4 Friend Finder

Die beschriebene Software trägt den Namen *Friend Finder* und wurde im Rahmen dieser Arbeit realisiert. Im folgenden wird auf die verwendeten Verfahren sowie Bibliotheken, die zur Realisierung notwendig waren, eingegangen und die Programmstruktur aufgezeigt.

4.1 Verwendete Verfahren und Bibliotheken

Friend Finder wurde so konzipiert, dass die graphische Darstellung ohne großen Aufwand von den restlichen Teilen der Software abgekoppelt und durch eine andere Art der Darstellung ersetzt werden kann.
Neben dem *Graphical User Interface (GUI)* besteht die Software aus drei unterschiedlichen Modulen. 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* zeigt den Kommunikationsaustausch von *Friend Finder*.

4.1.1 Grafisches Benutzeroberfläche

Um die Modularität zu wahren wird der gesammte Programmcode der Benutzeroberfläche in einer Datei zusammengefast. In dieser Datei sind alle Funktionen enthalten um die Oberflächenelement 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. Zur Darstellung der Karte wurden Daten des offenen Kartenprojekts *OpenStreetMap* [16] genutzt.

4.1.2 Versenden der Nachrichten

Der *Message Sender* kümmert sich um das Versenden von Nachrichten. Um das Versenden und Empfangen der Daten zu implementieren wurde *libircclient* [17] genutzt. Im ersten Schritt baut dieser eine Verbindung zum *IRC-Server* auf. Um eine Verbindung mit einem *IRC-Server* zu etablieren, 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, können nun Nachrichten versandt werden. Textnachrichten müssen vor dem Versenden in Blöcke aufgeteilt werden, da das genutze Verschlüsselungsverfahren *Blowfish* [8] maximal 64 Bit lange Zeichenkette verschlüsselt. Das verwendete Verfahren stammt aus

[16] OpenStreetMap http://www.openstreetmap.de/
[17] libircclien http://libircclient.sourceforge.net/

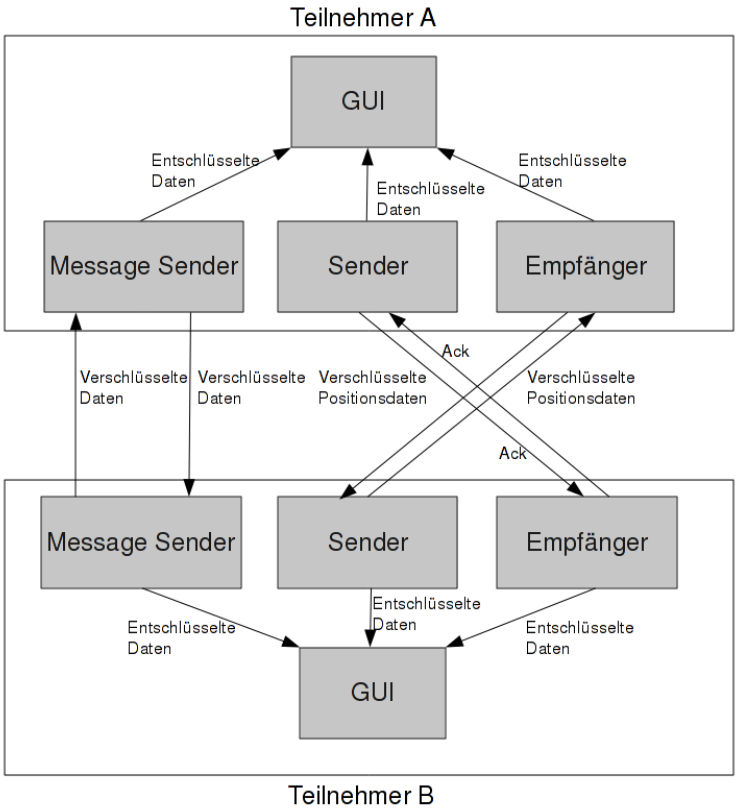

Abbildung 3: *Friend Finder* Nachrichtenaustausch

der *OpenSSL*[18] Bibliothek. Diese Implementierung wurde wurde Aufgrund der schnellen Verschlüsselungsrate sowie einfachen Implementierung gewählt. Da das *IRC*-Protokoll nicht alle Zeichen darstellen kann oder als Präfix vor einem Kommando nutzen werden alle versendeten Daten des Programmes in die *Base64* [19] Darstellung umgewandelt.
Wird nun von einer anderen Instanz des *Message Senders* eine Nachricht empfangen, so setzt er die Teilstücke zusammen. Dies geschieht solange, bis ein vom Nachrichtentext getrennt gesendetes Terminierungszeichen empfangen wird. Wurde dieses Zeichen empfangen, so gilt die Textnachricht als wiederhergestellt und wird an die Benutzeroberfläche weitergereicht.

4.1.3 Versenden der eigenen Position

Der *Sender* ist zuständig für das Versenden der Positionsdaten. Auch hier muss vor dem Versenden von Daten eine *IRC-Session* initialisiert werden. Der Ablauf beim Senden der Positionen erfolgt in einer vorgegebenen Reihenfolge. Zuerst wird der verschlüsselte Längengrad, danach

[18] OpenSSL http://www.openssl.org/
[19] Base64 RFC 4648

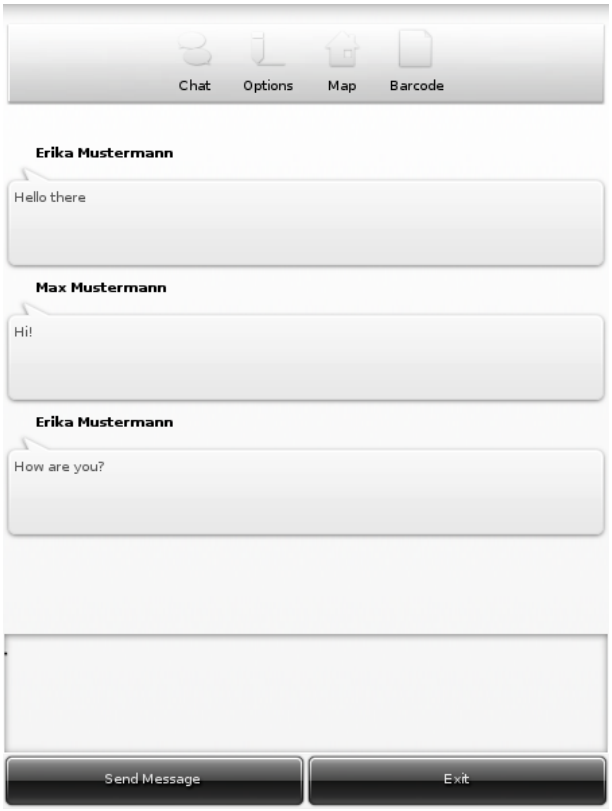

Abbildung 4: Versenden von Chatnachrichten

der verschlüsselte Breitengrade gesendet. Allerdings muss auch hier, wie beim Versenden der Textnachrichten, darauf geachtet werden dass maximal eine Zeichenkette der Länge von 64 Bit verschlüsselt wird. Somit ist es auch hier nötig Längen- und Breitengrad in zwei Teile aufzuteilen und getrennt zu versenden. An jedes Ende, dieser insgesamt vier Fragmente, wird ein zusätzlicher, jeweils unterschiedlicher Buchstaben angehängt. Da sich diese vier Buchstaben unterscheiden können die Positionsfragmente später identifiziert und eindeutig festgestellt werden ob es sich zum Beispiel um den ersten Teil einer *Longtitude*-Koordinate handelt. Somit werden für das Versenden von 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 für jedes Fragment an den *IRC-Channel* sendet. Kommt diese Bestätigung beim *Sender* an, so versendet dieser wieder ein *Latitude/Longtitude* Paar.

4.1.4 Empfangen von Positionen

Auch beim *Empfänger* muss im ersten Schritt eine *IRC-Session* initialisiert werden. Da mehrere Benutzer Positionsdaten senden können legt der *Empfänger* für jeden *Sender* einen Datensatz an. Wird nun ein Fragment der Positionsdaten empfangen, so kann der dies anhanden des Buch-

stabensuffix zuordnen. Sind alle Fragmente einer Position empfangen worden, so werden die benötigten Daten zur Visuallisierung weitergereicht und ein *Acknowledgement* gesendet. Dieses *Acknowledgement* beinhaltet, verschlüsselter Form, den Namen des *Senders* der Nachricht. Somit kann der *Sender* die für ihn bestimmten *Acknowledgements* zuordnen. Die folgende Abbildung zeigt die Darstellung einer Position eines anderen Benutzers.

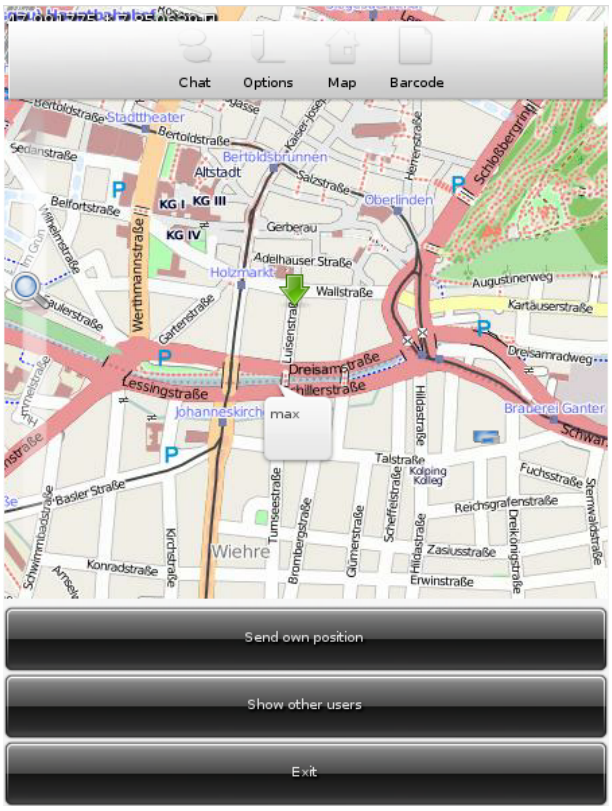

Abbildung 5: Ausgabe einer Benutzerposition

4.1.5 Erzeugen eines 2D-Barcodes

Um einen 2D-Barcode zu erstellen wird eine Zeichenkette benötigt. Aus dieser werden durch die Nutzung von *qrencode* [20] Bilddaten generiert. Diese werden im nächsten Schritt durch *libpng* [21] gerendert und auf das Speichermedium geschrieben. Dieses Bild wird dann durch die *GUI* geladen und ausgegeben. Die folgende Abbildung zeigt einen solchen erstellten Barcode, wie er von *Friend Finder* ausgegeben wird.

[20] libqrencode http://megaui.net/fukuchi/works/qrencode/index.en.html
[21] libpng http://www.libpng.org/

Abbildung 6: 2D-Barcode mit *Friend Finder*

4.2 Analyse

Beim versenden der Daten durch *Friend Finder* soll möglichst wenig Datenoverhead produziert und versendet werden. Unter Datenoverhead werden Hintergrunddaten gesehen, welche versendet werden um die Verbindung aufrecht zu erhalten oder um die Anzahl der verfügbaren Teilnehmer zu überprüfen.
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. Ein Interessanter Punkt stellt die Frage dar, wie sich das versendete Datenaufkommen im Vergleich zu einer Lösung verhält, welche die Daten an jeden Teilnehmer einzeln verschickt. Hier ist besonders von Interesse, ob der Datenoverhead den Vorteil eines *Broadcast*-Mediums wie ein *IRC*-Channel revidiert oder nicht.
Der *Traffic* wurde mit Hilfe des Programmes *Wireshark* [22] untersucht. Wie bereits erwähnt wird zum Versenden der Nachrichten das *IRC*-Protokoll verwendet. In dieser Testumgebung wurde die Software *IRCD-Hybrid* [23] genutzt. Der Server lief auf dem gleichen Computer wie der Client. Der Client hat sich in diesem Szenario über das *localhost* Interface mit dem Server verbunden.

[22] Wireshark http://www.wireshark.org/
[23] IRCD http://www.ircd-hybrid.org/

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 der Datenverkehr beim Versenden und Empfangen von Positionen genauer betrachtet. Alle folgenden Größen sich nur auf die Größe des Datenfeldes, exklusiv der Header.

4.2.1 Allgemeiner Datenverkehr

Der Allgemeine Hintergrundverkehr bei *Friend Finder* besteht zum Einen aus *Keep-Alive* Nachrichten, sowie der Anfrage des Clients nach aktiven Nutzern in den *Channels* in denen er selbst aktiv ist. Die *Keep-Alive* Nachrichten werden alle 30 Sekunden zwischen Server und Client ausgetauscht. Die Größe des Datenfeldes einer solchen Nachricht beträgt also 24 Byte. Das Datenfeld der Pakete welche von Server an Client gesendet werden hat die Größe von 44 Byte.
Die Anfragen nach den anderen Benutzern in einem *Channel* werden alle 60 Sekunden versandt. Die Größe der Pakete welche von Client zu Server gesandt werden, betragen hierbei 10 Bytes. Die Größe der Antwort des Servers hängt von der Anzahl der aktiven Benutzer innerhalb eines Channels ab. Für zwei Benutzer ergibt sich ein Datenvolumen von 193 Byte, wobei diese Größe auch Abhängig von der Länge der Benutzernamen sowie des Namens des*Channels* 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“. Dieses Paket hat ein Datenfeld der Größe von 99 Byte.
Die versendete Textnachricht hat im unverschlüsselten Format die Größe von elf Byte. Nach der Verschlüsselung werden beim Senden noch Informationen bezüglich *Channel* und der Empfänger der Nachricht in das zu versendende *IRC*-Paket geschrieben. Nach der *Base64*-Kodierung hat sich die Größe der Nachricht circa um den Faktor 9 vergrößert.
Wenn h die Größe des *TCP-Headers* und t die Anzahl der Zeichen der unverschlüsselten Nachricht ist, so ergibt sich die ungefähre Größe der zu versendenden Nachricht aus: $h + (t \cdot 9)$.

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. Bei der Messung wurden vier Verschiedene Pakete, mit jeweils unterschiedlichen versandten Positionen untersucht. Dabei betrug sich die Größe des Datenfeldes

um die 430 Byte. Die Anzahl der unverschlüsselten Zeichen, die für ein *Latitude/Longtitude*-Paar zu senden sind, beträgt 21 Zeichen. Jedes Zeichen ist Byte groß, womit sich in der Summe eine Größe von 21 Byte ergibt. Durch die Verschlüsselung, *Base64*-Kodierung sowie Zusatzinformationen vergrößert sich das Datenvolumen also um circa den Faktor zwanzig. 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 $h + (t \cdot 20)$. Hinzu kommt noch, das für jedes empfangene Positions-Fragmeint ein *Acknowledgement* gesendet wird. Die Größe der eines *Acknowledgment* Paketes beträgt zwischen 147 und 153 Byte. In einem solchen Paket werden vier Acknowledgments zusammengefasst.
Folglich kann Folgende Formel für den Datenverkehr pro versendeter Position, bei n Teilnehmern hergeleitet werden: $((h + (t \cdot 20)) + (4 \cdot a)) \cdot n$, wobei a die Größe eines Acknowledgement-Paketes ist und n die Anzahl der versandten Pakete repräsentiert.

4.2.4 Fazit der Auswertung

Die Hintergrunddaten welche vom *IRC*-Protokoll versandt werden ergeben einen geringen, in Kauf zu nehmenden Datenoverhead. Allerdings ist der große Vorteil von *IRC*, dass die *Channels* als *Broadcast*-Medium genutzt werden können. Diese Tatsache macht es möglich, Daten n Teilnehmer zugänglich zu machen und dabei diese nur einmal, über eine aktive Verbindung, zu senden. Berücksichtigt man dies, so fällt, der ohnehin geringe Datenoverhead, nicht mehr ins Gewicht. Würde man diese Daten über n getrennte Verbindungen an die Teilnehmer versenden, so müssten ebensoviele Verbindungen geöffnet werden und die Daten anstelle von einmal, n Mal versandt werden.
In der folgenden Graphik wird das Versenden der Daten über n getrennte Verbindungen, sowie die in*Friend Finder* implementierte Methode verglichen. Es wird angenommen das die versandten Positionsdaten eine Größe von 430 Byte, und ein *Acknowledgement* 150 Byte haben. Des weiteren wird angenommen dass alle Nutzer die Daten empfangen auch Positionsdaten senden und somit auch $4 \cdot n$ *Acknowledgements* versandt werden müssen. Daraus ergibt sich die Formel $(n \cdot 430) + (n \cdot 4 \cdot 150)$.

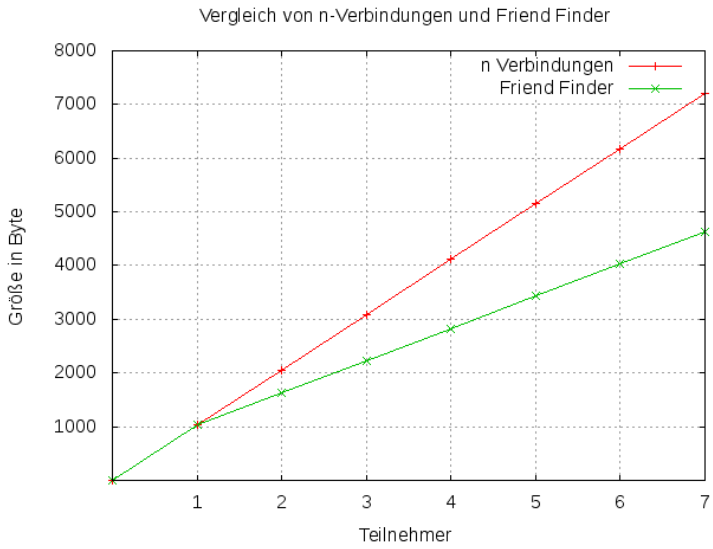

Abbildung 7: Vergleich von *n*-Verbindungen und *Friend Finder*

5 Fazit

Im Rahmen dieser Arbeit konnte gezeigt werden, dass es möglich ist einen Dienst zu entwickeln welcher die eigene Position und die anderer Teilnehmer anzeigt und dabei eine offene Struktur, in Form des *IRC*-Netzwerkes, nutzt. Dem Anwender ist somit die Möglichkeit gegeben die Kontrolle über die Nutzung seiner Daten zu wahren, da er seine versendeten Daten dank der genutzten Struktur stets einsehen kann. Des Weiteren kann er, durch Weitergabe der Schlüssel, bestimmen wer Einsicht in seine Positionsinformationen erlangen darf. Der Austausch der Schlüssel wurde mit der Methode der 2D-Barcodes so gelößt, dass dieser spontan und im mobilen Umfeld stattfinden kann, ohne dabei eine langfristige Vorausplanung zu benötigen.
Es wurde aufgezeigt, dass die Implementierung eines solchen Dienstes mit bereits vorhandenen Algorithmen und Bibliotheken sinvoll umsetzbar ist. In der Analyse des Datenverkehrs wurde aufgezeigt, dass das *IRC*-Netzwerk beim versenden von Daten nur einen geringen Datenoverhead produziert. Des weiteren eignet es sich für die Kommunikation mit mehreren Teilnehmern, da hier nur eine Verbindung geöffnet werden muss.

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

[1] Valentina Ciriani, Sabrina De Capitani di Vimercati, Sara Foresti, and Pierangela Samarati. k-anonymity. *Secure Data Management in Decentralized Systems*, pages 323–353, 2007.

[2] Matt Duckham and Lars Kulik. Location privacy and locationaware computing. *Dynamic and mobile GIS: investigating change in space and time*, pages 34–51, 2006.

[3] Marco Gruteser and Dirk Grunwald. Anonymous usage of location based services through spatial and temporal cloacking. *Proceedings of the 1st international conference on Mobile systems, applications and services*, pages 31–42, 2003.

[4] Hidetoshi Kido, Yutaka Yanagisawa, and Tetsuji Satoh. An anonymous communication technique using dummies for location-based services. *ICPS '05. Proceedings. International Conference on Pervasive Services*, pages 88–97, 2005.

[5] Günter Müller. 2009.

[6] J. Oikarinen and D. Reed. Rfc 1459 internet relay chat, 2003.

[7] Hanan Samet. The quadtree and related hierarchical data structures. *ACM Computing Surveys (CSUR)*, Volume 16:187–260, 1984.

[8] Bruce Schneier. Description of a new variable-length key, 64-bit block cipher (blowfish). *Fast Software Encryption, Cambridge Security Workshop Proceedings*, pages 191–204, 1994.

[9] Yaniv Shaked and Avishai Wool. Cracking the bluetooth pin. *Proceedings of the 3rd international conference on Mobile systems, applications, and services*, pages 39–50, 2005.

[10] Erik Sherman. Geocaching: hike and seek with your gps. *Computing Reviews*, Volume 46(2), 2005.

[11] Konstantin Welke and Klaus Rechert. Spontaneous privacy-aware location sharing, 2009.