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

Mobiler Friend Finder

Patrick Hornecker

Lehrstuhl für Kommunikationssysteme

Prof. Dr. Gerhard Schneider

betreut durch

Klaus Rechert

Technische 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 die fortschreitende 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. Aktuelle Modelle dieser 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 möglich, zum Beispiel eine zurückgelegte Strecke zu speichern oder den nächsten Supermarkt in der Nähe anzuzeigen. Des Weiteren können diese Positionsdaten für soziale Dienste genutzt werden, um die Nutzer auch anhand ihrer Position zu vernetzen. Es bieten sich also eine Vielzahl an Möglichkeiten, um diese Daten sinnvoll zu nutzen, allerdings kann die so entstehende Datenflut auch anderweitig genutzt werden. So kann aufgrund der Orte, an denen sich der Anwender befindet, gezielt Werbung platziert oder die Positionsdaten können über die Zeit gespeichert und Bewegungsprofile des Benutzers erstellt werden. Es ist somit 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, welche es ermöglicht mit anderen Benutzern die Positionen auszutauschen und anzuzeigen. Hierbei soll für den Benutzer Transparenz geboten werden, so dass er immer in der Lage ist, nachzuvollziehen was mit seinen Daten geschieht. Der Anwender soll durch Verschlüsselung bestimmen können, wer befähigt ist diese Informationen einzusehen.

2 Grundlagen

Location privacy wird von Duckham und Kulik (2006) 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 in der Lage sein, den Zeitpunkt, wie und in welchem Umfang Positionsdaten über ihn verbreitet werden, selbst festzulegen. Es stellt sich die Frage, wie es dem Benutzer ermöglicht wird, diese drei Punkte zu kontrollieren. Dabei soll er den Zeitpunkt frei bestimmen können, wann und auf welche Art und Weise 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. Anhand der Definiton von *location privacy* werden ebenfalls Anforderungen, die an ein Programm dieser Art gestellt werden analysiert und mögliche Ziele einer solchen Software beschrieben.

2.1 Aktuelle Entwicklungen

Nahezu alle aktuellen Smartphones sind mit dem *Global Positioning System (GPS)* ausgestattet. Aufgrund dessen, gibt es für die verschiedenen Betriebssysteme mittlerweile eine Reihe von Anwendungen, die Funktionalitäten rund um die eigene Position bieten. So gibt es Software 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 nutzen, auf einer Karte anzeigen zu lassen. Es besteht hierbei die Möglichkeit die eigene Position per *GPS* oder mit Hilfe von Daten der *GSM-Funkzellen* zu bestimmen.
Betrachtet man ein solches Programm unter den obigen Gesichtspunkten von Duckham und Kulik, so stellt man fest, dass der Anwender nur den Zeitpunkt zu dem die Positionsinformationen versendet werden, bestimmen kann. Nutzen Anwender die Software, 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 Ausgabe von Positionsdaten anderer. Er hat keinerlei Kontrolle darüber, was mit den Daten nach dem Absenden passiert, da diese an den Anbieter gesendet werden, wo sie dann durch ein ihm unbekanntes System an die anderen Nutzer weitergereicht werden. Somit ergibt sich eine Informationsasymmetrie, da der Betreiber mehr Informationen über die Verarbeitung der Daten des Anwenders besitzt, wohingegen der Benutzer nicht im Genauen weiß, wie mit seinen Daten verfahren wird. Der Benutzer kann also weder festlegen wie, noch in welchem Umfang er die

[1] Google Latitude http://www.google.com/intl/en_us/latitude/intro.html [Online; letzter Aufruf 11.02.2010]

Daten versenden möchte, da er keinen Einblick in diesen Teil der Strukturen der Institution, die den 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 das zum Versenden genutzte System offen zugänglich ist, wird eine zusätzliche Verschlüsselung dieser Daten benötigt, ansonsten hätte jedermann Zugang zu ihnen.
Zum Verschleiern der Position nutzen Kido u.a. [5] Datensätze die falsche Positionsangaben beinhalten. Diese werden an einen Dienst gesandt, welcher auf all diese Datensätze antwortet, egal ob diese richtige oder falsche 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 anderen 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, sowohl über den *3G*-Standard sowie per *WLAN* Daten zu übertragen.
Wenn nun ein sicherer Austausch von Positionsdaten erfolgen soll, so sind neben der Hardware, auch andere gegebene Rahmenbedingungen erforderlich. Da man durch Positionsdaten die aktuelle Position erfahren oder Bewegungsprofile erstellen kann, sollte der Zugang zu diesen nur dann erlaubt sein, wenn der Benutzer, der sie versendet, damit einverstanden ist. 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 versenden können. Es ist also notwendig, dass er in der Lage ist, die dafür benötigten Parameter sofort zu erstellen und weiterzugeben, um diesen Zeitpunkt frei wählen zu können.
Ist dies gegeben, werden die Daten im nächsten Schritt versendet. Auch hier ist erforderlich, dass der Anwender möglichst viel Kontrolle über seine Datensätze hat. Zur Wahrung dieser Kontrolle ist es notwendig, dass die Informationen mit einer möglichst transparenten und doch verlässlichen Methode verschickt werden. Zum weiteren Schutz der verschlüsselten Daten darf

zur Übertragung der Informationen nicht ein zentraler Knoten genutzt werden, sondern im optimalen Fall ein ganzes Netzwerk von solchen Knotenpunkten. Bei diesem Netzwerk sind allerdings alle Knotenpunkte 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 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, ohne dabei die Benutzerfreundlichkeit einzuschränken. Die Inhalte der Anwendung müssen also soweit abstrahiert werden, so dass ein Benutzer ohne Fachkenntnise alle Funktionen der Anwendung nutzen kann, ohne dass die obigen Punkte ausser Kraft treten.
Die Nutzung einer offenen Struktur zum Versenden der Daten ist von immenser Wichtigkeit, damit jeder Nutzer in diese Einsicht hat und sie frei genutzt werden kann. Es ist bei der Auswahl zu vermeiden, dass diese Struktur Restriktionen wie einen Benutzeraccount mit sich bringt. Ebenfalls sollte diese über ein Protokoll verfügen, das verlässlich, stabil und auch für langsame Netzwerke optimiert ist. Eine reibungslose Kommunikation kann somit garantiert werden.
Kommunikation zwischen verschiedenen Teilnehmern kann durch den Austausch von Chatnachrichten ermöglicht werden. Die Interaktion zwischen den Anwendern und somit der Nutzen des Dienstes kann hierdurch weiter gesteigert werden.
Durch die Nutzung einer für alle offenen Struktur ist es von Nöten, dass die Daten verschlüsselt werden. Da bei Verschlüsselungen der Austausch von Schlüsseln vorausgesetzt wird, ist die Nutzung eines Verfahrens notwendig, mit welchem der Austausch von Schlüsseln auf einfache Weise möglich ist. Da diese Software für mobile Geräte ausgelegt ist, ist der spontane Austausch der benötigten Schlüssel im Interesse des Benutzers, da der Anwender die Nutzung eines solchen Dienstes nicht immer im Vorraus planen kann und möchte. Wollen Benutzer zum Beispiel eine ihnen fremde Stadt erkunden und dabei diesen Dienst verwenden, so kann mit dieser Lösung, ohne Vorarbeit, eine Sitzung erstellt und die Paramater dafür untereinander ausgetauscht werden. Eine Planung im Voraus wird somit nicht benötigt. Im gleichen Zug ist zu garantieren, dass die Schlüssel während des Austauschs nicht von unbefugten Personen abgefangen werden können.
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, ist eine Visualisierung von Nöten. Somit ist die Fähigkeit andere Teilnehmer auf einer Karte anzuzeigen eine Funktion, welche als Teil dieser Anwendung gilt. Es wird also ein Format benötigt, 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 erreichbar 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 mehrfache Ausführung dieser vermieden wird. Zum Anderen werden möglichst viele Benutzer erreicht und es kann eine Kommunikation unter Besitzern von unterschiedlichen Typen von Mobiltelefonen stattfinden.
Ebenfalls von Bedeutung ist die Modularität einer Anwendung. Ist diese gegeben, so fällt das Austauschen oder Erweitern von Programmteilen auch zu einem späteren Zeitpunkt leicht. 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 ist man nun in der Lage, geeignete Verfahren und Protokolle sowohl für Kommunikation als auch für Verschlüsselung zu wählen. Da mit mehreren Benutzern oder auch mehreren Benutzergruppen kommuniziert werden 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. Um den Schlüsselaustausch einfach zu gestalten, wird eine symmetrische Verschlüsselung genutzt.
Da der Schlüsselaustausch spontan durchführbar sein soll, muss dieser zu jedem Zeitpunkt möglich sein, ohne dass dafür Vorbereitungen notwendig sind. 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 fotographiert und wieder in eine Zeichenkette umgewandelt werden. Da keine Kommunikation über einen unsicheren Kanal zwischen den Geräten stattfindet sind die Barcodes optimal zum spontanen Schlüsselaustausch geeignet, da der Schlüssel auf Daten diesem Wege nicht abgefangen werden kann.
Wurden die Schlüssel erfolgreich zwischen den Benutzern ausgetauscht, so sind diese in der Lage das Versenden der Informationen über eine geeignete Struktur zu beginnen. Aufgrund der Möglichkeit, dass 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 dieses Protokolls ist, wenn Daten an mehrere Benutzer gesendet werden sollen ist es nur notwendig diese einmal an einen *Channel* zu versenden. Jeder Benutzer in diesem *Channel* kann diese Daten daraufhin empfangen. Zusätzlich steht es jedem Benutzer frei, eigene *Channels* zu öffnen. Somit kann für

[2]QR Code http://www.denso-wave.com/qrcode/qrstandard-e.html [Online; letzter Aufruf 11.02.2010]

eine Sitzung, innerhalb derer Daten ausgetauscht werden, ein *Channel* eröffnet und nach Beendigung des Austausches wieder geschlossen werden. Es ergibt sich hier die Möglichkeit dynamisch *Channels* zu diesem Zweck zu nutzen.
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 werden. Diese verarbeiten die Daten im Anschluss, so dass diese als Position auf einer Karte ausgegeben werden können. Beim Versenden der Textnachrichten ist die Vorgehensweiße äquivalent.

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, findet die Auswahl aufgrund des Betriebssystemes statt. 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 dass die genutzten Bibliotheken ausgetauscht werden müssen.
Zusätzlich stellt sich auch die Frage der zu nutzenden Programmiersprache. Diese sollte von einer möglichst großen Menge an Betriebsystemen unterstützt werden. Somit ist es nicht nötig das Programm neu zu implementieren, falls es für eine neue Plattform portiert werden soll.

3.1 Betriebsysteme für mobile Geräte

Durch die Wahl eines Betriebsystemes wird schon indirekt eine Vorauswahl an nutzbaren Bibliotheken und Programmiersprachen getroffen. Im Folgenden werden fünf Betriebssysteme für mobile Plattformen vorgestellt und auf deren Portierungsmöglichkeiten eingegangen.

[3] POSIX http://standards.ieee.org/regauth/posix/ [Online; letzter Aufruf 16.02.2010]

3.1.1 Windows Mobile

Das von Microsoft entwickelte *Windows Mobile*[4] ist aktuell in der 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* mit dem Namen *CeGCC*[5], mit welchem in *C*/*C++* geschriebene Programme für diese Plattform übersetzt und portiert 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 Sprachen bei Nutzung des *SDKs* im Moment auf diese Eine beschränkt. Des Weiteren bietet *Google* ein *NDK* an, mit dessen 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 das zugrundeliegende System hat, *POSIX*-kompatible Software unter *WebOS* zu betreiben. Für dieses Betriebssystem existiert ein *SDK* welches *HTML5*, *CSS* und *Java* unterstütz. Ein weiteres *SDK*, welches im März 2010 veröffentlicht wird, soll die *C* und *C++* Entwicklung ermöglichen.

3.1.4 iPhone OS

Bei dem Betriebssystem *iPhoneOS*[9] handelt es sich um eine abgeänderte und angepasste Version von *MacOS*. Somit basiert auch der Kernel von *iPhoneOS* sowohl auf Teilen eines *BSD*-[10] sowie

[4] Windows Mobile http://www.microsoft.com/windowsmobile/de-de/default.mspx [Online; letzter Aufruf 25.01.2010]
[5] CeGCC http://cegcc.sourceforge.net/ [Online; letzter Aufruf 24.02.2010]
[6] Android http://www.android.com/ [Online; letzter Aufruf 25.01.2010]
[7] WebOS http://palmwebos.org/ [Online; letzter Aufruf 25.01.2010]
[8] PalmOS http://www.palm.com/ [Online; letzter Aufruf 25.01.2010]
[9] iPhoneOS http://www.apple.com/de/iphone/ [Online; letzter Aufruf 03.02.2010]
[10] BSD https://www.bsdwiki.de/ [Online; letzter Aufruf 24.02.2010]

Mach-Kernels[11]. 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 die geplante Anwendung stark einschränkt, da hier weitere Dienste im Hintergrund aktiv sein sollten.

3.1.5 Symbian OS

SymbianOS[12] ist ein Betriebssystem, welches vorzugsweise auf Geräten der Firma *Nokia* zum Einsatz kommt. Es existiert ein *SDK*, was neben *C*/*C++* auch noch weitere Sprachen wie zum Beispiel *Python* oder *Java* unterstützt. Mit dem *SDK* wird auch ein *Cross-Compiler* angeboten, welcher es ermöglicht Programme direkt zu portieren. Des Weiteren besitzt *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 somit auf einem solchen System nicht realisierbar wäre. *Android* hat zwar eine *C* Unterstützung, allerdings sind nur die mit dem *NDK* verfügbaren Bibliotheken zum momentanen Zeitpunkt stabil. Dies schränkt die Entwicklung stark ein und ist für das geplante somit Programm nicht geeignet.
Unter Nutzung des *CeGCC's* wird das Programm für *Windows Mobile* portiert werden. Aufgrund der Implementierung in *C* ist es möglich, den Quellcode für *WebOS* und *SymbianOS* zu übersetzen und im Anschluss auszuführen.

3.2 Softwaregrundlagen

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

[11] Mach http://www.cs.cmu.edu/afs/cs/project/mach/public/www/mach.html [Online; letzter Aufruf 24.02.2010]
[12] SymbianOS http://www.symbian.org/ [Online; letzter Aufruf 03.02.2010]

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, basierend 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. Die erste Version, der *arm-wince-mingw32ce*, bietet eine Menge an *Tools*, mit welchen native *Windows Mobile* Applikationen erstellt werden können. Hierzu wird eine Portierung der *GNU*[13] Enwicklungswerkzeuge aus dem *MinGW*[14] Projekt genutzt. Diese *Tools* übersetzen und verlinken Quellcode, um selbigen unter *Windows Mobile* ausführbar zu machen. Die andere Version stellt die Nutzung des *arm-cegcc* dar, mit welchem *POSIX* kompatibler Quellcode nach *Windows Mobile* portiert werden kann.

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 graphischen Elementen, die genutzt und frei angeordnet werden können.

Abbildung 1: Beispiele verschiedener *Elementary* Icons

[13]GNU http://www.gnu.org/ [Online; letzter Aufruf 03.02.2010]
[14]MinGW http://mingw.org/ [Online; letzter Aufruf 03.02.2010]
[15]Elementray http://www.enlightenment.org/ [Online; letzter Aufruf 20.01.2010]

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

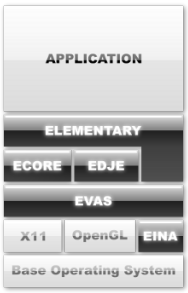

Abbildung 2: Aufbau von *Enlightenment*

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, welche Elemente an welchem Ort, in welcher Farbe und wie sichtbar sind. 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* für *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

Der *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*. In dieser Graphik sind die Module zweier Instanzen von *Friend Finder* abgebildet. Es ist ersichtlicht, dass die Kommunikation für Positionsdaten zwischen *Sender* und *Empfänger* stattfindet. Textnachrichten werden zwischen den *Message Sendern* ausgetauscht. Des Weiteren geben *Sender*, *Empfänger* und *Message Sender* die entschlüsselten Daten an die *GUI* weiter.

4.1.1 Graphische 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 Oberflächenelemente zu platzieren. Um die gewünschte Funktionalität der einzelnen Elemente zu realisieren, wurden 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

[16] OpenStreetMap http://www.openstreetmap.de/ [Online; letzter Aufruf 13.02.2010]
[17] libircclient http://libircclient.sourceforge.net/ [Online; letzter Aufruf 25.01.2010]

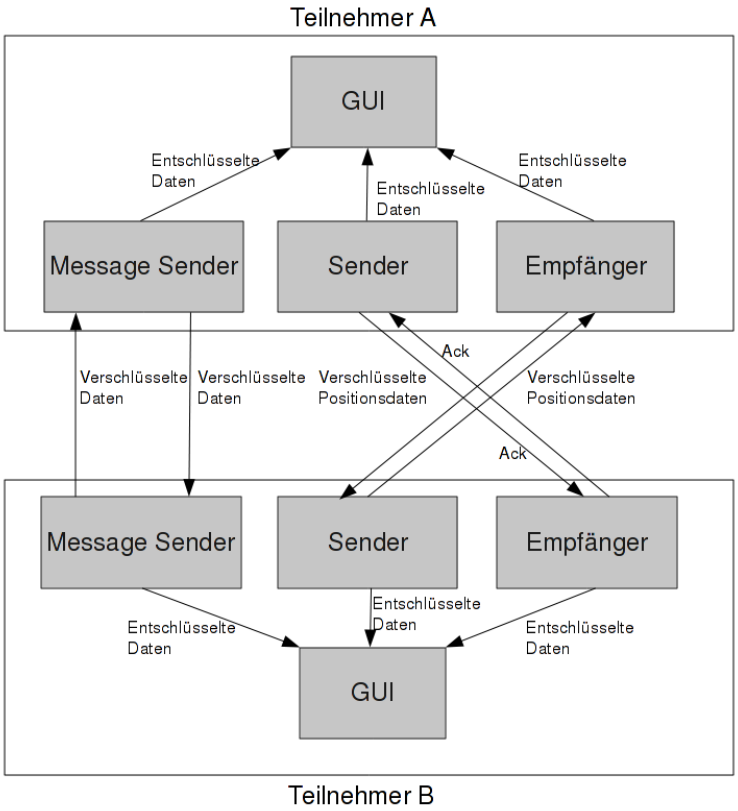

Abbildung 3: *Friend Finder* Nachrichtenaustausch

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 Nachrichten versandt werden. Textnachrichten müssen vor dem Versenden in Blöcke aufgeteilt werden, da das genutzte Verschlüsselungsverfahren *Blowfish* [8] maximal 64 Bit lange Zeichenketten verschlüsselt. Das verwendete Verfahren stammt aus der *OpenSSL*[18] Bibliothek. Diese Implementierung wurde aufgrund der schnellen Verschlüsselungsrate sowie einfachen Implementierungsmöglichkeiten gewählt. Da das *IRC*-Protokoll nicht alle Zeichen darstellen kann oder bestimmte Zeichen als Präfixe vor einem Kommando genutzt werden, werden alle versendeten Daten des Programmes in die *Base64*[4] 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. In Abbildung 4 ist die graphische Darstellung eines Nachrichtenaustausches abgebildet.

[18]OpenSSL http://www.openssl.org/ [Online; letzter Aufruf 25.01.2010]

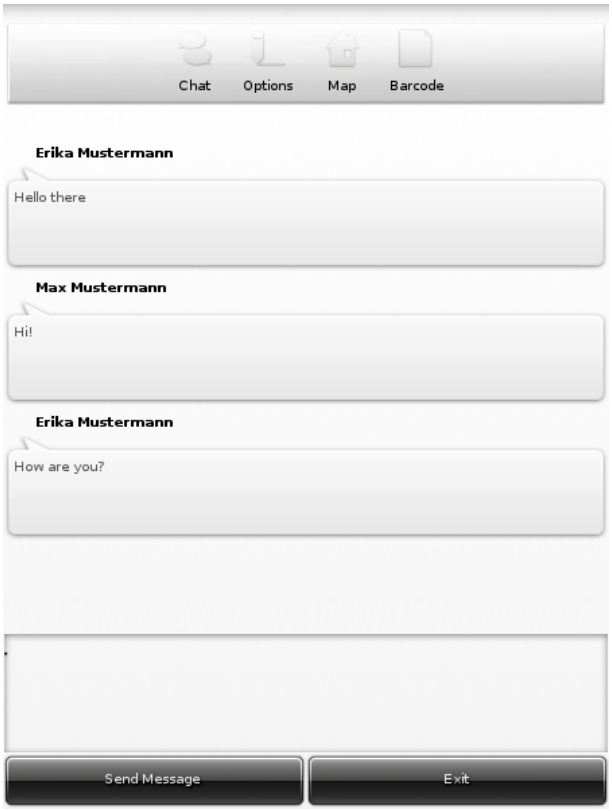

Abbildung 4: Versenden von Chatnachrichten

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 der verschlüsselte Breitengrade gesendet. Allerdings muss auch hier, wie beim Versenden der Textnachrichten, darauf geachtet werden, dass maximal eine Zeichenkette der Länge 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 Suffix angehängt. Durch diese Suffixe, können die Positionsfragmente später eindeutig zugeordnet werden. 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. Sollten diese *Acknowledgements* ausbleiben, so wartet der Sender eine längere Zeit und versendet die Daten erneut. Somit werden Daten in längeren Abständen versandt, wenn kein *Empfänger* aktiv sein sollte. Tritt nun wieder ein *Empfänger* dem *Channel* bei und sendet *Acknowledgements*, so erhöht der Sender

daraufhin wieder die Senderate. Diese Vorgehensweise tritt auch in Kraft, falls eine Bestätigung aufgrund von Übertragungsfehlern nur unvollständig ankommt. Sollte die Verbindung zum *IRC-Server* unterbrochen werden, so versucht sich der *Sender* in regelmäßigen Zeitabständen neu zu verbinden. Abbildung 5 zeigt das Versenden von Positionsdaten des *Senders* über einen *IRC-Channel* an den *Empfänger*. Dieser schickt im Anschluss auf gleichem Weg *Acknowledgements* an den *Sender*.

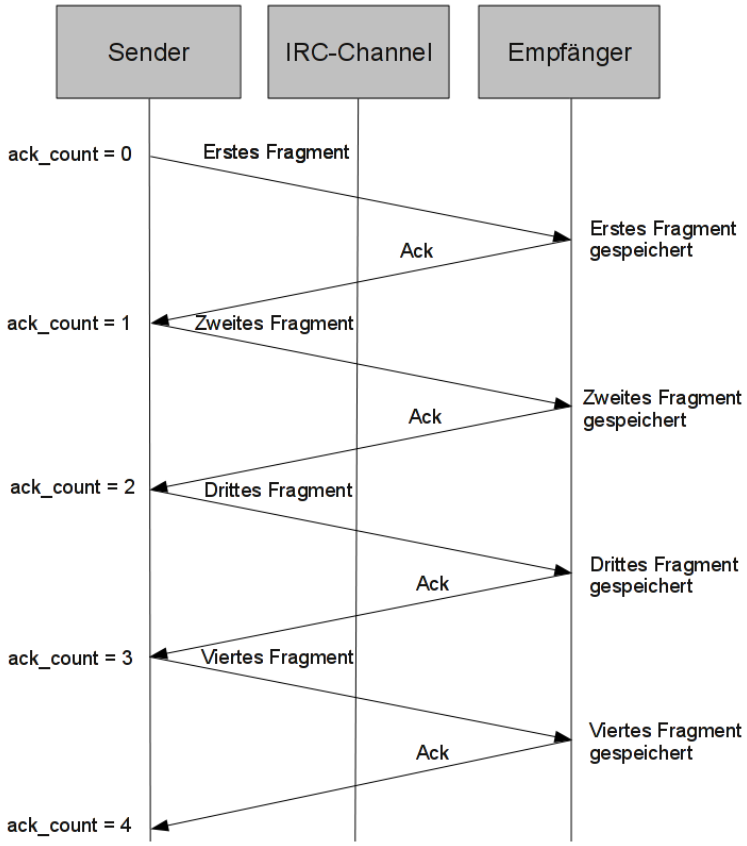

Abbildung 5: Versenden von Chatnachrichten

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 *Empfänger* dies anhanden des Buchstabensuffix zuordnen. Wurden alle Fragmente einer Position empfangen, so werden die benötigten Daten zur Visualisierung weitergereicht und ein *Acknowledgement* gesendet. Dieses *Acknowledgement* beinhaltet, in verschlüsselter Form, den Namen des *Senders* der Nachricht. Somit kann der *Sender* die für ihn bestimmten *Acknowledgements* zuordnen. Sollte der *Sender*

das Senden von Nachrichten einstellen, so wartet der *Empfänger* bis ein neuer *Sender* verfügbar ist. Sollte der Fall eintreten, dass die Positionsfragmente nicht in der vorgesehenen Reihenfolge ankommen, so stellt dies für den *Empfänger* kein Problem dar. Aufgrund des Suffixes, wird das Fragment gespeichert und zu einem Längen- oder Breitengrad vervollständigt, sobald der fehlende Teil vorhanden ist. Sollte ein Fragment nicht ankommen, so werden alle alten, bis dahin vorhandenen Fragmente verworfen. Dies geschieht, sobald der *Empfänger* registriert, dass ein Positionsfragment der gleichen Art schon vorhanden ist. Sollte ein Positionsfragment unvollständig ankommen, so wird es verworfen. Wird die Verbindung zum Server unterbrochen, so versucht der *Empfänger* sich neu zu verbinden. Abbildung 6 zeigt die Darstellung einer Position eines anderen Benutzers.

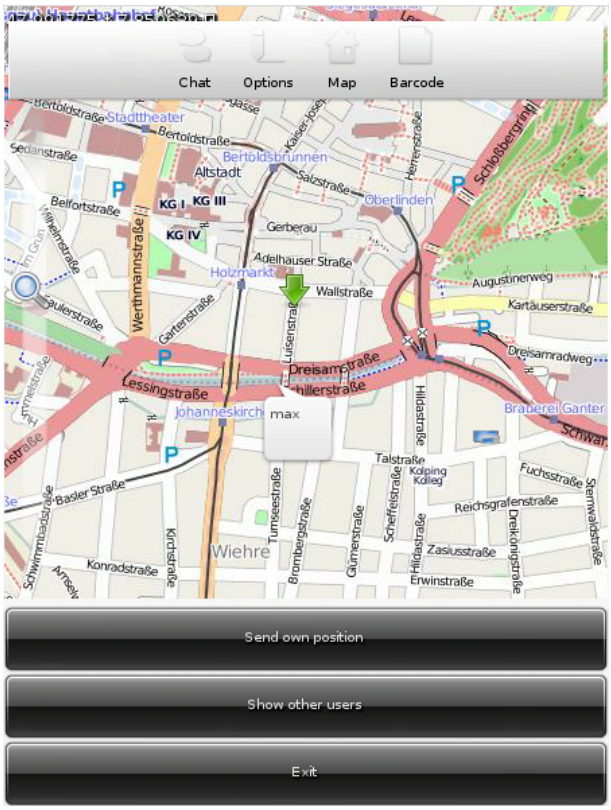

Abbildung 6: Ausgabe einer Benutzerposition

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* [19] Bilddaten generiert. Diese werden im nächsten Schritt durch *libpng*[20]

[19] libqrencode http://megaui.net/fukuchi/works/qrencode/index.en.html [Online; letzter Aufruf 11.02.2010]

gerendert und auf das Speichermedium geschrieben. Dieses Bild wird dann durch die *GUI* geladen und ausgegeben. Die Abbildung 7 zeigt einen solchen erstellten Barcode, wie er von *Friend Finder* ausgegeben wird.
Möchten Benutzer Schlüssel austauschen, so kann ein Barcode aus einer Zeichenkette erstellt werden. Die zugrunde liegende Zeichenkette kann zum Beispiel ein zuvor erstellter und auf dem Medium gespeicherter 1024 Bit Schlüssel, eine in *Friend Finder* eingegebene Zeichenkette oder ein Schlüssel speziell für eine Sitzung erstellt wurde, sein. Aus dieser kann nun ein Barcode erstellt werden und andere Teilnehmer können diesen ihrerseits fotographieren, in die ursprüngliche Zeichenkette umwandeln und als Schlüssel nutzen. Der Vorteil, der durch die Wahl dieser Methode ensteht, ist dass Schlüssel ausgetauscht werden können, wenn Nutzer sich durch Zufall treffen oder den Dienst spontan nutzen wollen, ohne dabei schon im Voraus Schlüssel erstellen zu müssen.

Abbildung 7: 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* versteht man Hintergrunddaten, welche versendet

werden um die Verbindung aufrecht zu erhalten oder um die Anzahl der verfügbaren Teilnehmer zu überprüfen. Diese Daten besitzen allerdings keinen Informationsgehalt für den Anwender.
Im folgenden Teil wird der erzeugte Datenverkehr von *Friend Finder* analysiert. Das Hauptaugenmerk wird hierbei auf die Paketgröße, 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*[21] untersucht. Wie bereits erwähnt, wird zum Versenden der Nachrichten das *IRC*-Protokoll verwendet. In dieser Testumgebung wurde die Software *IRCD-Hybrid*[22] genutzt, welche auf dem gleichen Computer wie der Client lief. Der Client hat sich in diesem Szenario 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 und Empfangen von Nachrichten. Im letzten Teil dieser Analyse wird der Datenverkehr beim Versenden und Empfangen von Positionen genauer betrachtet. Alle folgenden Größen beziehen sich nur auf die Größe des Datenfeldes, exklusive 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 einer solchen Nachricht von Client zu Server beträgt 24 Byte. Das Antwortpaket, welches vom Server an den Client gesendet wird, hat eine Größe von 44 Byte.
Anfragen nach den anderen Benutzern innerhalb eines *Channel* werden alle 60 Sekunden versandt. Die Größe der Pakete welche von Client zu Server versandt werden, betragen hierbei zehn 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* und Servers ist.

4.2.2 Versenden und Empfangen von Nachrichten

Um das Versenden von Textnachrichten zu evaluieren wurde „Hello World“ als Testnachricht benutzt. Der in *Friend Finder* genutzte *Blockcipher* teilt den Satz „Hello World“ in zwei Teile auf: „Hello “ und „World“. Das resultierende Paket hat eine Größe von 99 Byte.

[21] Wireshark http://www.wireshark.org/ [Online; letzter Aufruf 27.01.2010]
[22] IRCD http://www.ircd-hybrid.org/ [Online; letzter Aufruf 27.01.2010]

Die versendete Textnachricht plus Terminierungszeichen hat im unverschlüsselten Format die Größe von 24 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 Verschlüsselung, *Base64*-Kodierung, sowie hinzufügen von Zusatzinformationen hat die Größe der Nachricht circa um den Faktor vier zugenommen.
Wenn h die Größe des *TCP-Headers* und t die Anzahl der Zeichen einer unverschlüsselten Nachricht repräsentiert, so ergibt sich die ungefähre Größe der zu versendenden Nachricht aus: $h + (t \cdot 4)$.

4.2.3 Versenden und Empfangen von Positionen

Positionsdaten werden beim Sender aufgeteilt und mit vier unterschiedlichen Nachrichten versandt. Bei der Messung wurden vier verschiedene Pakete, mit jeweils unterschiedlichen versandten Positionen untersucht. Dabei beträgt die Größe des Datenfeldes im Mittel 140 Byte. Die Gesamtgröße der unverschlüsselten Nachrichten um ein *Latitude*/*Longtitude*-Paar zu versenden beträgt 32 Byte. Durch die Verschlüsselung, *Base64*-Kodierung sowie Zusatzinformationen vergrößert sich das Datenvolumen also um circa den Faktor vier. Wenn h die Größe des *TCP-Headers* und t die Anzahl der Zeichen der unverschlüsselten Nachricht darstellt, ergibt sich die Größe der versendeten Nachricht circa durch $h + (t \cdot 4)$. Hinzu kommt, dass für jedes empfangene Positions-Fragment ein *Acknowledgement* gesendet wird. Die Größe eines *Acknowledgement*-Paketes beträgt zwischen 110 und 120 Byte. In einem solchen Paket werden vier *Acknowledgements* zusammengefasst.
Daraus kann folgende Formel für den Datenverkehr pro versendeter Position bei n Teilnehmern hergeleitet werden: $((h + (t \cdot 4)) + (4 \cdot a)) \cdot n$, wobei a die Größe eines *Acknowledgements* 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*. Der große Vorteil von *IRC* ist, 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 Abbildung 8 wird das Versenden der Daten über n getrennte Verbindungen, sowie die in *Friend Finder* implementierte Methode, zu einem beliebigen Zeitpunkt t verglichen. Es wird angenommen, dass die versandten Positionsdaten eine Größe von 140 Byte und vier *Acknowledgements* von 120 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 $(n \cdot 140) + (n \cdot 120)$, welche als Grundlage für die Abbildung 8 dient.
In Abbildung 9 ist der Hintergrund Traffic von *Friend Finder*, zu einem beliebigen Zeitpunkt t, abgetragen. Die Größe wird anhand der aktiven Teilnehmer ausgegeben. Es wird angenommen, dass eine Nachricht mit Benutzerinformationen eines *Channels*, für n Benutzer die Größe von $70 + (60 \cdot n)$ hat. Die Größe von 70 Byte stellt hierbei den Server plus den *Channel*-Namen dar. Zudem wird angenommen, dass in diesem Fall die Information zu jedem Benutzer eine Größe von 60 Byte hat. Eine *Keep-Alive* Nachricht hat die Größe von 136 Byte, da diese Nachrichten in dem Zeitraum in dem einmal die Benutzerinformationen ausgetauscht werden, zweimal versandt werden. Dieses Datenvolumen setzt sich aus der Nachricht von Client zu Server, sowie der Antwortnachricht des Servers an den Client zusammen. Die Gesamtgröße der *Keep-Alive* Nachrichten für n Teilnehmer wird durch $136 \cdot n$ errechnet. Die eingezeichnete Summe repräsentiert die Formel $(136 \cdot n) + (70 + (60 \cdot n)$, also die Summe aus *Keep-Alive* Nachrichten sowie versendeten Benutzerinformationen.
Es fällt also auf, dass beim *IRC*-Protokoll trotz Hintergrunddaten, bei steigender Teilnehmerzahl weniger Datenvolumen benötigt wird. Würde eine Lösung mit n Verbindungen gewählt werden, so muss noch bedacht werden, dass auf die versendeten Daten Hintergrunddaten, multipliziert mit der Anzahl der Teilnehmer, addiert werden müssten. Im Gegensatz hierzu müssen diese Werte beim *IRC*-Protokoll nur einmalig addiert werden.

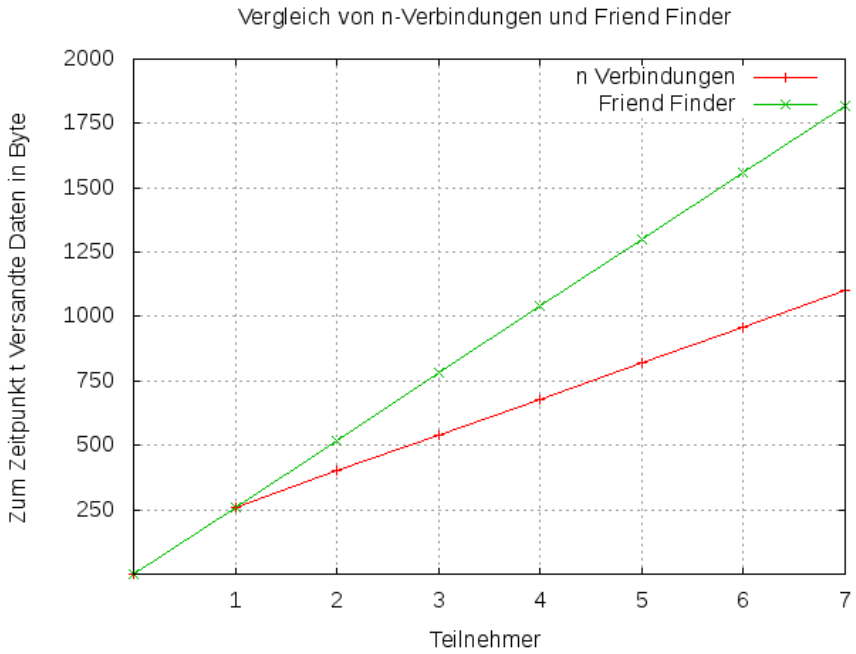

Abbildung 8: Vergleich des versandten Datenvolumen, anhand von unterschiedlichen Teilnehmerzahlen, von n-Verbindungen und *Friend Finder* zu einem beliebigen Zeitpunkt t.

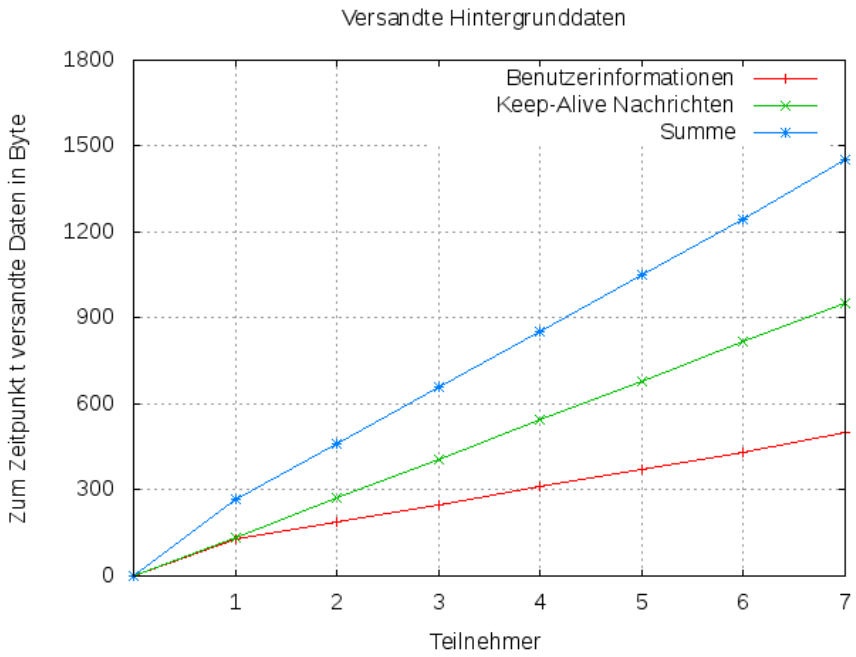

Abbildung 9: Versandte Hintergrunddaten von *Friend Finder*, anhand von unterschiedlichen Teilnehmerzahlen, zu einem beliebigen Zeitpunkt t.

5 Fazit

Im Rahmen dieser Arbeit wurde ein Dienst zum Austauschen von Positionen konzpiert und implementiert. Ein Schwerpunkt lag hierbei auf der Nutzung einer offenen, einsehbaren Struktur um die Daten zu versenden. Aufgrund dieser Offenheit ist es notwendig, die Daten zu verschlüsseln. Ebenfalls wurde es ermöglicht die Parameter, welche für eine Sitzung notwendig sind, spontan zu erstellen und auszutauschen.
Als offene Struktur wurde das bereits vorhandene *IRC*-Protokoll, sowie dessen existente Netzwerkstruktur genutzt. Zur Verschlüsselung der zu versendenden Daten wurde ein symmetrischer Algorithmus genutzt. Um Schlüssel auszutauschen, werden aus diesen 2D-Barcodes erstellt, welche von anderen Nutzern fotographiert und wiederrum in einen Schlüssel umgewandelt werden können.
Es wurde gezeigt, dass das *IRC*-Protokoll aufgrund des geringen *Datenoverhead*, sowie den *Channels* die sich als *Broadcast-Medium* nutzen lassen, für das Versenden solcher Daten an viele Benutzer bestens geeignet ist. Des Weiteren ist die Nutzung von 2D-Barcodes zur Weitergabe der Schlüssel gut geeignet, da diese im mobilen Umfeld einen schnellen, unkomplizierten und sicheren Austausch dieser Parameter ermöglichen.
Im Anschluss an diese Arbeit stellt sich die Frage, ob es möglich ist eine noch effizientere Struktur als das *IRC*-Protokoll zu entwickeln, welche eine ebenso offene Struktur bietet, aber speziell für den Austausch von Positionsinformationen geeignet ist. Ebenfalls stellt sich die Frage, ob es möglich ist, eine noch größere Anzahl an Betriebssysteme zu unterstützen und das Programm für diese zu portieren. Ein weiterer Punkt ist die Frage, ob noch andere Möglichkeiten existieren, um im mobilen Umfeld Schlüssel unter möglichst einfachen Umständen und ohne Vorarbeit auszutauschen.

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:
Zuerst muss eine aktuelle Version des *CeGCC's* heruntergeladen und installiert werden. Die benötigten Dateien können auf der Projekthomepage[23] heruntergeladen werden. Für dieses Tutorial wird der *mingw32ce* für den *ARM*-Prozessortyp benötigt. Nach dem Herunterladen muss dieser in das passende Systemverzeichniss entpackt werden.
Bevor man mit dem nächsten Schritt fortfahren kann, müssen noch ein paar benötigte Pakete aus dem *Ubuntu-Repository* installiert werden.

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

Als nächstes muss ein Verzeichniss angelegt werden, in welchem die für Windows Mobile kompilierten Dateien gespeichert werden können. Des Weiteren muss noch eine Datei angelegt werden, in welcher die Pfade zun genutzten Kompiler liegen und welche dann einmalig exportiert werden müssen. Dies ist notwendig, damit die zum Portieren benötigten *Header*-Dateien, Bibliotheken und ausführbare Dateien auch vom Betriebssystem gefunden werden. Diese Datei wird im folgenden mit *mingw32ce.env* benannt. Das Anlegen dieser Datei geschieht mit Hilfe des Befehls `touch mingw32ce.env`. Nun müssen noch folgende Einträge in der Datei *mingw32ce.env* hinzugefügt 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
```

[23] http://cegcc.sourceforge.net/

```
                -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* exportiert werden. Die Variablen existieren nur in den *Shells*, in denen sie exportiert wurden.
Unter den Variablen *CEGCC_PATH* und *MINGW32CE_PATH* ist der Pfad zum Verzeichnis des *cegcc*, beziehungsweise des *mingw32ce* Kompilers einzutragen. Unter *WINCE_PATH* muss der Pfad zum Verzeichnis in dem die kompilierten Dateien gespeichert werden sollen, eingetragen werden. Mit *PATH* werden die *Binaries* des *CeGCC's*, in den systemweiten Pfad der ausführbaren Dateien aufgenommen. Des Weiteren werden unter *CPPFLAGS* die *include*-Pfade und unter *LDFLAGS* die Pfade zu den benötigten Bibliotheken abgelegt. *LD_LIBRARY_PATH* zeigt auf den Ordner in welchem ausführbare Dateien liegen, die zum Kompilieren benötigt werden. *PKG_CONFIG_PATH* zeigt schliesslich noch auf den Ordner, der die Paketinformationen der installierten Dateien beinhaltet. Das Exportieren dieser Werte geschieht mit dem Aufruf `source <Pfad-zu-der-Datei>/mingw32ce.env`.
Nun muss noch ein Ordner angelegt werden, in welchem der *Enlightenment Source-Code* gespeichert werden kann. Nach dem Wechseln in selbigen kann mit dem nächsten Schritt fortgefahren werden.

Evil

Zuerst muss der Quellcode von *Evil* aus dem *SVN* heruntergeladen werden. Das Herunterladen wird mit `svn co http://svn.enlightenment.org/svn/e/trunk/evil` ausgeführt. Nachdem alle Dateien erfolgreich heruntergeladen wurden, muss falls nicht schon geschehen, die Datei mit den *Umgebungsvaribalen* exportiert werden. Nachdem dies durchgeführt wurde, kann nun das Konfigurationsskript mit dem Aufruf `./autogen.sh --prefix=$WINCE_PATH --host=arm-mingw32ce` gestartet werden.
Durch das Ausführen des Skriptes mit diesen Parametern wird der Installationspfad auf den Wert der Variable *WINCE_PATH* gesetzt und der *mingw32ce*-Kompiler als Kompiler festgelegt.
Nachdem dieses Skript erfolgreich ausgeführt wurde, kann im nächsten Schritt das Programm durch Ausführen von `make` erstellt und im Anschluss durch `make install` installiert werden. Nach diesem Schritt sollte *Evil* erfolgreich im Zielordner, der in der Variable *WINCE_PATH* festgelegt wurde, installiert worden sein.

Eina

Hier ist es ebenfalls nötig die Dateien aus dem Entwickler-Repository durch `svn co http://svn.enlightenment.org/svn/e/trunk/eina` herunterzuladen. Danach wird das *autogen.sh* Skript durch

`./autogen.sh --prefix=$WINCE_PATH --host=arm-mingw32ce --disable-pthread` aufgerufen.
Es werden bei diesem Aufruf die gleichen Parameter wie bei *Evil* übergeben. Hinzu kommt noch *–disable-pthread*. Mit diesem Parameter wird die Nutzung der *POSIX* Bibliothek *libpthread* deaktiviert, da diese auf dem Zielsystem nicht unterstützt wird.
Nachdem das Skript ausgeführt wurde, wird das Programm nun mit `make` erstellt und im Zielverzeichnis mit `make install` installiert.

Eet

Bevor man *Eet* erstellen kann, muss man noch vier *tar-Archive* im Verzeichnis, welches in der Variable *WINCE_PATH* gespeichert wurde, entpacken. Diese Archive können unter den Links, welche in Anhang 2 zu finden sind, heruntergeladen werden. Im Anschluss müssen diese in das *WINCE_PATH*-Verzeichnis kopiert und entpackt werden. Nun kann der Quellcode für *Eet* durch Ausführen von `svn co http://svn.enlightenment.org/svn/e/trunk/eet` heruntergeladen werden. Nachdem die Dateien vorhanden sind, muss das *autogen.sh*-Skript durch `./autogen.sh --prefix =$WINCE_PATH --host=arm-mingw32ce` aufgerufen und ausgeführt werden. Im Anschluss wird mit `make` kompiliert und mit `make install` installiert.

Embryo

Der Erste Schritt ist das Herunterladen des Programmcodes unter der Verwendung von `svn co http://svn.enlightenment.org/svn/e/trunk/embryo`. Nachdem die Dateien heruntergeladen wurden, muss durch `./autogen.sh --prefix=$WINCE_PATH --host=arm-mingw32ce` das *autogen.sh* Skript aufgerufen werden. Im Anschluss mit `make` kompilieren und mit `make install` installieren.

Evas

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

```
cp $WINCE_PATH/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
```

Die Paketinformationen müssen allerdings noch bearbeitet werden. Dazu werden folgende drei Dateien, *freetype2.pc*, *libpng.pc* und *libpng12.pc* benötigt. Diese werden daraufolgend mit einem beliebigen Editor geöffnet und allen drei Dateien der Wert der Variable *prefix* auf *$WINCE_PATH*

gesetzt. Nachdem dies durchgeführt wurde kann nun der Quellcode von *Evas* durch Ausführen von `svn co http://svn.enlightenment.org/svn/e/trunk/evas` heruntergeladen werden. Nun wird `./autogen.sh --prefix=$WINCE_PATH --host=arm-mingw32ce --disable-async-events` ausgeführt. Im nächsten Schritt muss das Programm mit `make` kompiliert werden. Sollte hierbei die Datei *ft2build.h* nicht gefunden werden, so muss diese an die richtige Stelle kopiert werden. Ursprünglich ist diese Datei unter `$WINCE_PATH/freetype-2.3.7-dev/include/freetype2/ft2build.h` zu finden. Es kann allerdings vorkommen das sie in diesem Ordner nicht gefunden wird. Um dies zu beheben muss *ft2build.h* eine Ordnerebene nach oben kopiert werden. Dies geschieht mit dem Aufrufen von `cp $WINCE_PATH/freetype-2.3.7-dev/include/freetype2/ft2build.h $WINCE_PATH/freetype-2.3.7-dev/include .`
Um einen weiteren Fehler von vorneherein zu umgehen, muss noch der Pfad eines eingebundenen Headers in *ft2build.h* abgeändert werden. Hierzu öffnet man *ft2build.h* mit einem beliebigen Editor und ändert `#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 das *DirectX-SDK* herunterladen und *ddraw.h* in die Verzeichnisse */opt/cegcc/arm-cegcc/include/w32api/* und */opt/mingw32ce/arm-mingw32ce/include/* kopiert werden.

Ecore

Um *Ecore* zu Erstellen muss zuerst 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 *Ecore* erstellt werden. Dazu wird das *autogen.sh*-Skript durch den Aufruf von `./autogen.sh --prefix=$WINCE_PATH --host=arm-mingw32ce --disable-pthread` gestartet.
Nachdem das Skript erfolgreich ausgeführt wurde, können nun die gleichen zwei Schritte wie bei den vorhergegangenen Programmen ausgeführt werden. Der Kompilierungsvorgang wird durch `make` gestartet. Im Anschluss kann mit `make install` installiert werden.

Edje

Auch hier müssen zuerst die Dateien durch Aufruf von `svn co http://svn.enlightenment.org /svn/e/trunk/embryo` heruntergeladen werden. Nachdem die Dateien heruntergeladen wurden, muss das Skript mit `./autogen.sh --prefix=$WINCE_PATH --host=arm-mingw32ce` aufgerufen. Die letzten beiden Schritte sind kompilieren und installieren durch `make` und `make install`.

Elementary

Zuerst ist es notwendig den Quellcode und die benötigte Daten herunterzuladen. Dies geschieht durch den Aufruf von `svn co http://svn.enlightenment.org/svn/e/trunk/TMP/st/elementary`. Nun muss das *autogen.sh*-Skript mit `./autogen.sh --prefix=$WINCE_PATH --host=arm-mingw32ce --with-edje-cc=$WINCE_PATH/bin/edje_cc` gestartet und ausgeführt werden. Um zwei Fehlern vorzubeugen, welche beim Erstellen der Test-Files von *elementary* auftreten, sollte man in der Datei *Makefile.am* im Ordner *<Pfad-zu-Elementary>/src/bin/* alle Vorkommnisse von *test_file-selector.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 mit `make` und `make install` erstellt und installiert werden.

Weitere Schritte

Im Anschluss an das Erstellen dieser Programme muss nun noch ein Skript in *WINCE_PATH* angelegt und dessen Zugriffsrechte abgeändert werden. Das Anlegen des Skripts erfolgt durch `touch efl_zip.sh`. Die Zugriffsrechte werden durch `chmod 774 efl_zip.sh` abgeändert. In dieses Skript wird nun der Code kopiert, 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* zusammengefasst. Möchte man nun noch eigene Anwendungen hinzufügen, so muss man diese nur in diesen *efl* Ordner hinzufügen und das Skript aus Anhang 4 erneut ausführen. 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] V. Ciriani, S. De Capitani di Vimercati, S. Foresti, and P. Samarati. k-Anonymity. *Secure Data Management in Decentralized Systems*, pages 323–353, 2007.

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

[3] M. Gruteser and D. 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] S. Josefsson. RFC 4648 The Base16, Base32, and Base64 Data Encodings. 2006.

[5] H. Kido, Y. Yanagisawa, and T. Satoh. An anonymous communication technique using dummies for location-based services. *ICPS '05. Proceedings. International Conference on Pervasive Services*, pages 88–97, 2005.

[6] J. Oikarinen and D. Reed. RFC 1459 Internet Relay Chat. 2003.

[7] H. Samet. The Quadtree and Related Hierarchical Data Structures. *ACM Computing Surveys (CSUR)*, Volume 16:187–260, 1984.

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

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

[11] K. Welke and K. Rechert. Spontaneous Privacy-Aware Location Sharing. *In proceedings of the 4th International Conference on Pervasive Computing and Applications (ICPCA 2009)*, 2009.