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

Mobiler, persönlicher Assistent

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

In der heutigen, vernetzten Gesellschaft wird es den Leuten relativ einfach gemacht sich untereinander, auch über große Entfernungen, auf dem digitalen Weg zu verständigen. Die Spannweite dieser Vernetzung reicht von E-Mails, Personal Messanger bis hin zu sozialen Netzwerken wie zum Beispiel *facebook*, *StudiVZ* oder *twitter*.

Seit einiger Zeit wird diese Vernetzung um eine neue Möglichkeit erweitert. Die Masse an neuen Modellen und der Erfolg von *Smartphones* ermöglicht es dem Nutzer immer und so gut wie überall über das Internet verfügbar zu sein. So existieren schon Unmengen an Kommunikationssoftware und Applikationen die speziell für diese Mobilen Geräte entwickelt wurden.

Mit dem Aufkommen dieser neuen und mobilen Internetplattform muss auch hier wieder die Frage nach der Sicherheit von Daten und Persönlichen Informationen gestellt werden. So gelten für *Smartphones* andere Vorraussetzungen wie für Desktop Computer. Der Energieverbrauch sollte möglichst gering gehalten werden, um den Akku nicht allzuschnell zu entladen. Es muss auch berücksichtigt werden, dass die Mobilen Geräte nicht an die Leistungsfähigkeit regulärer Computer heranreichen und somit nicht die aufwändigsten Verfahren gewählt werden können.

Der Inhalt dieser Bachelor-Arbeit behandelt eben dieses Thema: Das verschlüsselte versenden von Informationen auf *Smartphones* und des weiteren die Möglichkeiten von *location awareness*. Für diesen Zweck wurde ein Programm namens *"Friend Finder"* implementiert, welche einfache Chat Nachrichten sowie die eigene Position versenden kann. Diese Positionen können dann von anderen Benutzern auf ihrem Mobilen Gerät angezeigt werden. Des weiteren beinhaltet diese Arbeit eine Analyse des Datenverkehrs, welcher durch diese Software erzeugt wird. Es werden auch andere Möglichkeiten von *location awareness* und Verschlüsselung aufgezeigt und besprochen.

Ein weiterer Punkt der aktuellen Generation der *Smartphones* ist, das es verschiedene Betriebssysteme gibt. Diese unterscheiden sich je nach Hersteller. Diese Ausarbeitung beinhaltet auch ein kurzes Tutorial, um das Softwarepacket *Enlightenment*[efl] von einer Linux-Plattform nach Windows Mobile zu konvertieren.

2 Technische Grundlagen

Eine Applikation für mobile Geräte, welche das Ziel der Kommunikation zwischen Benutzern sowie deren Vernetzung hat, sollte auf möglichst vielen Plattformen ausführbar sein. Somit wäre es möglich viele Nutzer zu erreichen und auch die Kommunikation zwischen einem Besitzer eines *iPhones* sowie dem Besitzer eines *PalmPre* wäre sichergestellt, ohne die Software mehrere Male zu implementiern. Des weiteren ist es auch von Interesse, ob andere Programme und Bibliotheken auf den jeweiligen Systemen ausführbar sind. Grundlegend sind Betriebsysteme interessant, die über einen *POSIX-Layer* verfügen. Dieser Layer stellt eine Schnittstelle zwischen Anwendungen und Betriebssytem dar. Somit können Anwendungen die auf einem *Linux*-System entwickelt wurden ohne weiteres auf ein anderes, *POSIX* kompatibles System, portiert werden. Auch die Frage der unterstützten Programmiersprachen stellt sich, da das Programm nicht ständig neu implementiert werden soll, wenn es auf ein neues Gerät portiert wird.

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

2.1 Betriebsysteme für mobile Geräte

Im folgenden werden die fünf bekanntesten Betriebssysteme für mobile Geräte kurz vorgestellt und auf die Portierungsmöglichkeiten der einzelnen Systeme eingegangen.

2.1.1 Windows Mobile

Der wohl bekannteste Vertreter ist *Windows Mobile*. Die aktuelle Version 6.5 wurde von Microsoft auch *Windows Phone* betitelt.Das gesamte Betriebssystem basiert auf der *Windows Win32 API* und lässt ähnlichkeiten zu den Desktop-Varianten der Windows-Familie erkennen. *Windows Phone* besitzt keinen *POSIX Layer*, allerdings existiert ein *Cross-Compiler* names *CeGCC* [CeGCC], mit welchem Programme die in *C*/*C++* geschrieben wurden für diese Plattform kompiliert werden können.

2.1.2 Android

Bei *Android* [Android] handelt es sich um ein neueres Betriebssystem für Smart Phones. Das von Google entwickelte System setzt auf einen Linux-Kernel der Version 2.6 auf. Dieser Kernel kümmert sich um die Prozess- und Speicherverwaltung, Kommunikation sowie um die Hardwareabstraktion.

Zum implementieren von Anwendungen stellt Google eigens ein eigenes *SDK* bereit. Dieses greift allerdings nur auf *Java*-Bibliotheken zurück, womit sich die nutzbaren Sprachen im moment eben auf diese beschränken. Somit können Programme die in *C* oder *C++* geschrieben wurden nicht portiert werden. Auch wenn man einen passenden *Cross-Compiler* nutzt ist die Portierung nicht immer möglich, da Google die *libc*-Bibliothek (unter *Android* nun *Bionic* genannt) an mobile Geräte angepasst und verändert hat.

Durch diese starken Einschränkungen und dem fehlende *POSIX Layer* ist es somit nicht möglich Programme, welche unter Linux oder in *C*/*C++* entwickelt wurden, für dieses System zu kompilieren.

2.1.3 WebOS

WebOS [WebOS] gehört nicht zu den weit verbeiteten Betriebsystemen, allerdings wird es hier aufgeführt, da *Enlightenment* portiert werden kann. Das System wurde von *Palm* als Nachfolger von *PalmOS* entwickelt. Momentan ist das System nur auf zwei Geräten zu finden: Auf dem *Palm Pre* und dem *Palm Pixi*.

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

2.1.4 iPhone OS

Bei *iPhoneOS* [iPhoneOS] handelt es sich um eine portierte Version von MacOS. Es wurde eigens für das iPhone entwickelt. Auch für dieses System existiert ein *SDK*, welches allerdings nur die Sprache *Objective-C* unterstützt. Des weiteren fehlt auch eine Unterstützung des *POSIX Layers*. Der größte Kritikpunkt an diesem System dürfte allerdings das fehlen von *Multitasking*-Unterstützung sein. Somit ist es nicht möglich zwei Anwendungen parallel auszuführen, was

gerade *location awareness* Anwendungen stark einschränkt, da hier häufig weitere Dienste im Hintergrund aktiv sein sollten.

2.1.5 Symbian OS

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

2.1.6 Zielplattform

Als primäre Zielplattform für diese Arbeit wurde *Windows Mobile* gewählt. Im ersten Schritt wurde das benutzen von *iPhone OS* und *Android* ausgeschlossen. Aufgrund ihrer Restriktionen, wie den fehlenden *POSIX Layer* und Einschränkungen der *SDK's* sowie die unterstützten Sprachen sind Programme, welche für diese Plattform entwickelt wurden nur auf diesen nutzbar. Diese Tatsache und die nicht vorhandene *Multitasking*-Unterstützung des *iPhones* machen diese zwei Geräte für diese Arbeit uninteressant.

Aufgrund der weiten Verbreitung, sowie der vorhandenen Tools ist die Wahl für dieses Projekt auf *Windows Mobile* gefallen. So werden zum einen viele Benutzer erreicht und zum anderen kann die Software mit geringem Aufwand nach *Symbian OS* und *WebOS* portiert werden und ist somit bei richtiger Implementierung auf mehreren Plattformen ausführbar.

Aufgrund der Entscheidung, *Windows Mobile* zu nutzen, wird als *Cross-Compiler* der *CeGCC* verwendet. Mit desen Hilfe können in einer Linx Umgebung die für *Windows Mobile* benötigten Bibliotheken und ausführbaren Dateien erstellt werden.

2.2 Softwaregrundlagen

Um Software, welche unter Linux entwickelt wurde, nach *Windows Mobile* zu portieren werden bestimmte Softwarelösungen vorrausgesetzt, welche im folgenden kurz erläutert werden.

2.2.1 CeGCC

Aufgrund der Entscheidung, *Windows Mobile* zu nutzen, wird als *Cross-Compiler* der *CeGCC* verwendet. Bei *CeGCC* handelt es sich um ein *Open-Source* Projekt, bassierend auf dem GCC. Mit desen Hilfe können in einer Linux Umgebung die für *Windows Mobile* benötigten Bibliotheken und ausführbaren Dateien erstellt werden.

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

2.2.2 Enlightenment

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

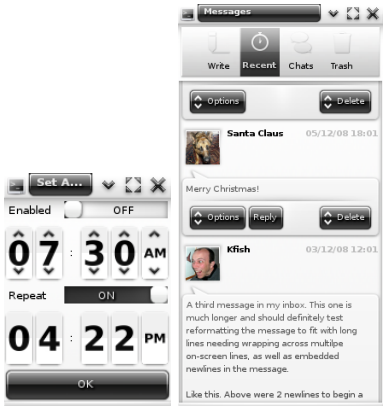

Abbildung 1: Beispiele verschiedener *Elementary* Icons

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

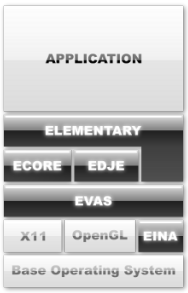

Abbildung 2: Aufbau von *Enlightenment*

Bei *Ecore* handelt es sich um eine *library* welche das serialisieren von mehreren Programmteilen ermöglicht und für den Betrieb auf mobilen Geräten optimiert wurde. *Edje* ist eine komplexe 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 wird. 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 ist eine genaue Anweisung zu finden, um *Enlightenment* für *Windows Mobile* zu erstellen.

3 Friend Finder

So gut wie alle Geräte der heutigen Generation von Smart Phones besitzen einen GPS Empfänger. Dieser kann auf unterschiedliche Art und Weiße genutzt werden. Von einfachem bestimmen der aktuellen Position, über Routing bis hin zu Freizeitaktivitäten wie *Geocaching*. Eine weitere, interessante Möglichkeit wäre es, sich Freunde anzeigen zu lassen, die sich in einem bestimmten Radius um die eigene Position befinden. Mit diesen könnte man dann, ähnlich wie mit einem Chat Programm, Nachrichten austauschen.
Da es unter Datenschutzaspekten aber nicht wünschenswert ist das jeder Dritte die Position anderer ermitteln kann, sollten die Daten verschlüsselt versendet werden. Ansonsten könnte es zu Szenarien, wie das erstellen eines Bewegungsprofiles der Benutzer, kommen. Es muss somit gewährleistet sein, dass die gesendeten Daten nur dann lesbar sind, wenn der Nutzer dem einwilligt.

Mit einer solchem Programm wäre es zum Beispiel möglich, sich mit Freunden zum Essen zu verabreden, wenn diese sich in der Nähe der eigenen Position befinden. Der Nutzer sollte einfach nur den Radius wählen, innerhald dessen Personen sichtbar sind, und könnte diese dann per verschlüsselter Textnachricht anschreiben.
Eine andere Möglichkeit bestünde darin, dass man auf bestimmten Veranstaltungen wissen möchte wer teilnimmt. Dabei stellt der Nutzer allerdings fest das eine ihm bekannte Person die gleiche Software nutzt, aber für ihn nicht sichtbar ist. Der Benutzer muss nun die Möglichkeit haben dieser Person ohne aufwand den für die Verschlüsselung genutzen Schlüssel zu übergeben.

Eine ähnliche Software wurde schon im Paper *Spontaneous Privacy-Aware Location Sharing* [Klaus Rechert, 2009] beschrieben und implementiert. Allerdings lag bei dieser Arbeit der Focus auf einem effizienten Protokoll zur verschlüsselten Positionsübertragung. Der Aspekt, dass Nutzer nur innerhalb eines begrenzten Radius angezeigt werden, wurde hier nicht berücksichtigt. Auch die Software *SmokeScreen* [Landon P. Cox, 2007] beschäftigt sich mit versenden privater Informationen innerhalb von Gruppen die der Nutzer festlegt. Allerdings bewegt sich *SmokeScreen* nich im Rahmen von *location awareness*. Somit sollte die Software ein effizientes Protokoll, eine sichere Verschlüsselung sowie die Möglichkeit der Einstellung durch den Anwender verbinden.

3.1 Anforderungsanalyse

Anhand der oben erstellten Szenarien sollte diese Software also folgende Funktionen besitzen:

- Versenden von Nachrichten
- Versenden der eigenen Position

- Anzeigen der Position von anderen Teilnehmern
- Erstellen eines 2D-Barcodes

3.1.1 Eigene Position senden

Um den Standort anderer Nutzer zu sehen, muss das Programm in der Lage sein, auf einer Karte deren Position anzuzeigen. Damit für andere Nutzer die eigene Position sichtbar ist, muss diese in einem gängigen Format versendet werden. Hierfür fiel die Wahl auf das Standart Positions Format *Latitude/Longtitude*. Um Datensicherheit zu garantieren müssen diese Positionsdaten in verschlüsselter Form versendet werden.

3.1.2 Position anderer Teilnehmer anzeigen

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

3.1.3 Nachrichten versenden und empfangen

Damit man mit anderen Nutzern, welche sich in der Nähe der eigenen Position befinden, auch Kommunizieren kann, muss die Software in der Lage sein Textnachrichten zu versenden und zu empfangen. Auch soll die Datensicherheit garantiert sein. Der Datenverkehr muss also auch bei dieser Funktion in verschlüsselter Form stattfinden.

3.1.4 2D-Barcode

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

3.2 Verwendete Verfahren und Bibliotheken

Die oben beschriebene Software wurde mit fast all diesen Funktionen, ausser das abfotographiern und umwandeln der 2D-Barcodes in einen Schlüssel, im Rahmen dieser Arbeit implementiert und hört auf den Namen *Friend Finder*. Im folgenden Abschnitt wird die Implementierung und Funktionsweise der einzelnen Funktionen erläutert, sowie die zugrundeliegenden Bibliotheken vorgestellt.

Friend Finder wurde so konzipiert, dass die Graphische Darstellung ohne großen Aufwand vom den restlichen Teilen der Software abgekoppelt und durch eine andere, darstellende Bibliothek ersetzt werden kann. Somit könnte man *Enlightenment* durch eine andere Art der Darstellung austauschen, ohne dabei die Funktionalität der zugrunde liegenden Komponenten zu zerstören. Da das ver- und entschlüsseln der Daten möglichst wenig Rechenaufwand erzeugen und der Schlüsselaustausch nicht zu kompliziert sein soll, nutzt das Programm ein symmetrisches Verschlüsselungsverfahren.
Abbildung 3 zeigt den Kommunikationsaustausch von *Friend Finder*. Der *Message Sender* ist für das Versenden und Empfangen der Textnachrichten zuständig, *Sender* sendet die eigene Position, *Receiver* empfängt die Positionen der anderen Nutzer und sendet Acknowledgements an die teilnehmenden *Sender*. Alle drei Teile geben ihre empfangenen Daten an die *GUI* weiter, welche sie mit Hilfe von *Enlightenment* ausgibt.

Friend Finder ist an sich in fünf Submodule aufgeteilt:

1. Graphische Benutzeroberfläche
2. Versenden von Textnachrichten
3. Versenden der eigenen Position
4. Empfangen der eigenen Position
5. Erstellen eines Barcodes

3.2.1 Grafisches Benutzeroberfläche

Zum erstellen der Oberfläche wurde *Enlightenment* verwendet. Diese Bibliothek stellt alle benötigten Funktionen bereit und bietet eine Fülle an vordefinierten Oberflächenelement. Der gesammte Programmcode der Benutzeroberfläche wurde in einer Datei zusammengefast (*gui.c*). Diese Tatsache vereinfacht das Erhalten der Modularität, da einfach nur diese Datei durch eine andere ersetzen werden muss um einen anderen Typ von Oberfläche zu benutzen.
In der der Datei *gui.c* sind alle Funktionen enthalten um die Oberflächenelement zu erzeugen und zu platzieren. Um die gewünschte Funktionalität der einzelnen Elemente zu realisieren wurden auch die Aufrufe der benötigten Funktionen aus anderen Modulen in dieser Datei implementiert. Wie schon erwähnt, wurde die Graphische Nutzeroberfläche mit Hilfe von Bibliotheken aus dem *Elementary*-Paket realisiert.

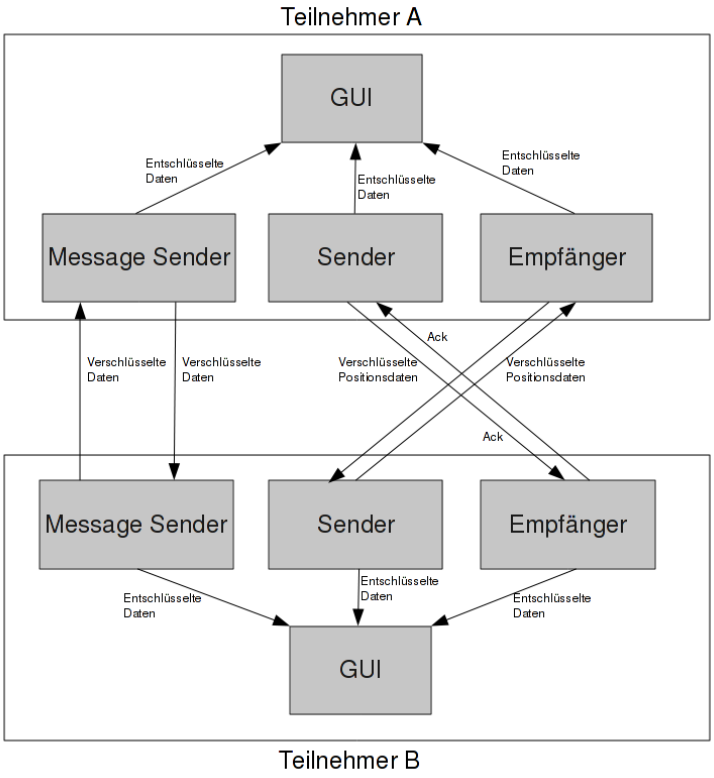

Abbildung 3: *Friend Finder* Nachrichtenaustausch

3.2.2 Versenden der Nachrichten

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

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

dies genau acht ASCII-Zeichen entspricht, werden alle zu sendenden Nachrichten in Blöcke der Größe acht aufgeteilt, versendet und beim Empfänger wieder zusammengesetzt.
Ein weiterer wichtiger Unterschied zu den Modulen Senden und Empfangen von *GPS*-Positionen ist die Tatsache, dass bei diesem Programmteil Sender und Empfänger in der gleichen Datei implementiert wurden. Der Grund hierfür ist, dass man hier nicht zwischen mehreren Sendern oder Empfängern unterscheiden muss, und diese zwei Teile hier somit nicht komplett getrennt voneinander arbeiten müssen.

Um Nachrichten zu versenden wurde für dieses Projekt die *IRC-Client Bibliothek*[libircclient] verwendet. Diese *library* bietet verschiedene Funktionen um eine Verbindung mit einem *IRC-Server* zu erstellen und Nachrichten an diesen zu senden, sowie eingehende Nachrichten zu empfangen.
Zur Ver- und Entschlüsselung der gesendeten Nachrichten, sowie der Positionsdaten wird die Bibliothek des *OpenSSL-Projekts*[OpenSSL], namens *libcrypto*, verwendet. Hierzu werden die Daten mit dem *Blowfish-Algorithmus* verschlüsselt. Bei diesem Algorithmus handelt es sich um ein symmetrisches Verfahren, bei welchem alle Teilnehmer den gleichen privaten Schlüssel zum ver- sowie entschlüsseln nutzen.

3.2.3 Versenden der eigenen Position

Der benötigte Programmcode zum Versenden der eigenen Position ist in der Datei *sender.c* zu finden. Auch hier muss zuerst eine *IRC-Session* initialisiert werden um danach die Position zu versenden. Der Ablauf beim Senden der Positionen erfolgt in einer vorgegebenen Reihenfolge. Zuerst wird der verschlüsselte Längengrad, danach der verschlüsselte Breitengrade gesendet. Daraufhin werden solange keine Daten mehr gesendet, bis der Empfänger eine Bestätigung an den *IRC-Kanal* sendet. Diese Bestätigung ist unverschlüssel. Kommt dieses *Acknowledgement* beim Sender an, so versendet dieser wieder ein *Latitude/Longtitude Paar*.

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

3.2.4 Empfangen der eigenen Position

noch nicht final....empfänger empfängt nachricht, ordnet sie, leitet sie zum zeichnen weiter....grob gesehen

3.2.5 Erzeugen eines 2D-Barcodes

Die Datei *barcode.c* beinhaltet die Funktionen zum erstellen eines 2D-Barcodes. Hierzu wird die Funktion "*generate_barcode(char* key)*", mit einer Zeichenkette als Übergabeparameter, aufgerufen. Aus dieser Zeichenkette wird dann ein Barcode erstellt, welcher im darauf folgenden Schritt als *.png* Datei auf das Speichermedium geschrieben wird.

3.3 Analyse

Das Ziel war es mit *Friend Finder* Daten verschlüsselt zu übertragen. Es soll dabei ein möglichst geringer Berechnungsaufwand entstehen um die Daten zu verschlüsseln, sowie möglichst wenig Datenoverhead produziert und versendet werden. Dies ist notwendig, damit das Programm möglichst lange auf dem mobilen Gerät ausgeführt werden kann und nicht schon nach kurzer Zeit der Akku versagt.

Im folgenden Teil wird der erzeugte Datenverkehr von *Friend Finder* analysiert. Ein Hauptaugenmerkt wird hierbei vor allem auf die Packetgröße, sowie die Menge der versendeten Datenpakete geworfen. Der *Traffic* wurde mit Hilfe des Programmes *Wireshark* [Wireshark] untersucht.

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

3.3.1 Allgemeiner Datenverkehr

Wie bereits erwähnt wird zum Versenden der Nachrichten das *IRC-Protokoll* [IRC] verwendet. Dieses Protokoll basiert *TCP/IP* und führt somit bei Verbindungsaufbau einen *TCP-Handshake* durch. Des weiteren werden während der gesamten Verbindungsdauer *ACK*- oder *NACK-Pakete* für jede Nachrichten versendet.

Während einer bestehenden Verbindung zu einem *IRC-Server* schicken sich Server und Client fortlaufend Nachrichten. So sendet der Client zum Beispiel immer eine "WhoÄnfrage an den Server, welcher dann mit allen im Channel eingelogten Nutzern antwortet.

Wird eine bestehnde Verbindung beendet, so entsteht auch hier wieder Datenverkehr, da *TCP* auch beim Beenden einer Verbindung ein eigenes Verfahren nutz. Bei diesem wird ein Paket versendet welches angibt dass die Verbindung nun beendet wird. Dieses Paket wird auch wieder bestätigt. Diese beiden Pakete werden von beiden Seiten der Kommunikation versandt.

3.3.2 Versenden und Empfangen von Nachrichten

Um das Versenden von Nachrichten zu evaluieren wurde ”Hello Worldäls Testnachricht benutzt. Der *Blockcipher* von *Friend Finder* teilt den Satz ”Hello Worldïn zwei Teile auf: ”Hello ünd ”World”. Diese werden dann von *TCP* aufgrund der Fenstergröße in ein Paket gepackt. Das gesamte Paket hat die größe von 81 Bytes, wobei der *TCP-Header* 32 Bytes und der *IP-Header* 20 Bytes groß ist. Somit haben die Daten eine Größe von insgesamt 29 Bytes.
Beachtet man dass ein *char* in *C* die Größe von einem Byte hat und der Beispielsatz aus elf Zeichen besteht, so ist dieser unverschlüsselt 11 Byte groß. Somit vergrößern sich die Daten nach der Verschlüsselung um circa den Faktor 2,6.
Wenn h die Größe des *TCP-Headers* und t die Anzahl der Zeichen der unverschlüsselten Nachricht ist, so ergibt sie die Länge der zu versendenden Nachricht aus: $h + (t * 2,6)$.

3.3.3 Versenden und Empfangen von Positionen

3.3.4 Fazit der Auswertung

Zur Analyse des Allgemeinen Datenverkehrs ist zu sagen dass Aufgrund der Tatsache dass das *IRC-Protokoll* auf *TCP/IP* bassiert, eine großer Overhead an Paketen versandt wird. Somit werde wesentlich mehr Pakete als nur die benötigten Daten verschickt. Hinzu kommen noch Pakete welche zur ständigen Kommunikation zwischen Server und Client ausgetauscht werden.

Um den den von *TCP* generierten Datenoverhead zu minimieren, wäre auch das versenden von Daten über *UDP* interessant. Dies gilt weniger für das versenden der Textnachrichten sondern eher für das versenden der Positionen. Hier werden, wenn mehrere User aktiv sind, ständig Positionsdaten und Acknowledgements zwischen Server und Clients ausgetauscht. Somit könnte durch die Nutzung von *UDP* hier einiges an versendeten Packeten gespart werden.

4 Ausblick

Die Software *Friend Finder* zeigt nur einen recht einfachen Ansatz auf, um Daten sicher zu versenden und die Position von anderen Benutzern aufzuzeigen. Im nun folgenden Teil dieser Arbeit werden Ausblicke auf weitere Möglichkeiten und Ansätze, für Nachrichtenverschlüsselung und *location awareness*, aufgezeigt.

4.1 Kryptographische Verfahren auf Mobilen Plattformen

In diesem Absatz werden mögliche kryptographische Verfahren für Smart Phones behandelt. So wird prinzipiell zwischen zwei verschiedenen Arten der Verschlüsselung unterschieden. Zum einen existieren die sogenanten symmetrischen und zum anderen die asymmetrischen Verfahren. Bei ersterem existiert nur ein *private-key* welcher von allen Teilnehmern zum ver- sowie entschlüsseln genutzt wird. Bei den asymmetrischen Verfahren sind mehrere Schlüssel nötig. Hierbei werden Daten mit einem öffentlichen Schlüsel verschlüsselt und können nur mit dem dazu passenden privaten Schlüssel wieder entschlüsselt werden.

Im folgenden werden verschiedene Algorithmen und deren Tauglichkeit vorgestellt. Des weiteren wird auch auf Möglichkeiten eingegangen, um Schlüssel zu verteilen.

4.1.1 Symmetrische Verschlüsselungsverfahren

Bei der Klasse der symmetrischen Verschlüsselungsverfahren können theoretisch alle bekannten Verfahren genutzt werden. Natürlich stellt sich hier die Frage der Güte der Verschlüsselung, die die einzelnen Algorithmen bieten. Das eigentliche Problem bei der Nutzung dieser Verfahren ist, wie man den privaten Schlüssel an die anderen Nutzer weitergeben kann, ohne ihn dabei für dritte zugänglich zu machen.

Als naiven Ansatz könnte man hierzu den Austausch per *Bluetooth* erwägen. Allerdings gilt *Bluetooth* nicht als sicheres Verfahren um Daten zu übertragen. Somit eignet sich dieses Art des Austauschens nicht, da nicht garantiert ist das Dritte mithören und den *private key* erhalten.

Eine weitere Möglichkeit wäre einen 2D-Barcode aus einer Zeichenkette zu erstellen. Diesen könnte man dann auf dem Display eines mobilen Gerätes ausgeben. Andere Nutzer könnten dann den Barcode fotographieren und wiederrum in eine Zeichenkette umwandeln. Diese Zeichenkette könnte dann für beide Kommunikationspartner als privater Schlüssel genutzt werden.
Was allerdings gegen diese Methode spricht ist die Tatsache, dass sich zwei oder mehrere Nutzer

treffen müssen um diese Daten auszutauschen. Somit verliert dieser Ansatz den großen Nachteil, dass die Mobilität ein Stück weit verloren geht, da man sich vor beginn der sicheren Kommunikation erst treffen muss.
Man könnte auch einfach die SD-Karten der mobilen Geräte mit einem oder mehreren gespeicherten, privaten Schlüsseln mit anderen Nutzern austauschen. Aber auch hier verliert man stark an Mobilität.

Man könnte auch eine Implementierung von *Kerberos* [Kerberos] für mobile Geräte erwägen. Bei diesem Verfahren übernimmt eine vertrauenswürdige dritte Partei die Authentifizierung. Diese wird von einem geschützten *Kerberos* durchgeführt. Wenn ein Client sich gegenüber einem Server authentifizieren möchte, muss er sich beim *Kerberos* Server anmelden. Dieser verifiziert dann seine eigene Identität, sowie Client gegenüber Server und Server gegenüber Client.
Möchte man nun diesen Dienst auf Smart Phones erweitern, so muss man einen *Kerberos* Client für die jeweilige Plattform entwickeln.

4.1.2 Asymmetrische Verschlüsselungsverfahren

Der Vorteil von asymmetrischen Kryptographiesystemen gegenüber den symmetrischen ist, dass man nur den eigenen privaten Schlüssel geheim halten muss während der öffentliche Schlüssel frei zugänglich ist. Bei symmetrischen Verfahren muss man alle privaten Schlüssel speichern und geheim halten. Durch diesen Vorteil der *Public-Key* Verfahren ist das Verteilen der einzelenen Schlüssel wesentlich einfacher. Das Problem von vorgetäuschten öffentlichen Schlüssen durch dritte kann mit einer vertrauenswürdigen Zertifizierungsstelle oder einem *Web of Trust* stark eingegrenzt werden.

Das Prinzip des *Web of Trust* macht sich das *Pretty Good Privacy (PGP)* Verfahren zu nutze. Dieses Verfahren wurde von der Firma *PGP* in Form von *PGP Mobile* [PGPmobile] für Smart Phones realisiert. Das *PGP*-Verfahren ermöglicht es Daten zu verschlüsseln, signieren oder sowohl zu verschlüsseln als auch zu signieren. Bei einem signierten und Verschlüsselten Datenpaket wendet der Sender zuerst ein Hash-Verfahren auf die zu verschlüsselnden Daten an. Im nächsten Schritt wird der private Schlüssels des Senders auf eben diesen Hash angewendet um eine Signatur zu erstellen. Soll die Nachricht nun verschlüsselt werden, so wird das unverschlüsselte Datenpaket und die Signatur komprimiert und mit einem zufällig generierten Schlüssel verschlüsselt. Da dieser zufällige Schlüssel nur einmal gültig ist, wird er asymmetrisch mit dem öffentlichen Schlüssel des Empfängers verschlüsselt und an die Nachricht angehängt. Nun kann die gesamte Nachricht an den Empfänger gesendet werden.
Der Empfänger kann nun mit Hilfe seines private Schlüssels den angehängten Schlüssel dieser Sitzung entschlüsseln und mit desen Hilfe die komprimierte Nachricht wiederherstellen.
Für dieses Verfahren existiert auch eine *Open Source* Variante namens *OpenPGP* [openPGP].

Mit dieser offenen Implementierung wäre es möglich ein solches Verfahren auch als freie Variante zu implementieren.
Der Nachteil des genannten Verfahrens ist mit Sicherheit der große Overhead an Daten versendet werden muss, sowie die Maße an benötigten Operationen um einen Klartext zu verschlüsseln. Dies tritt vor allem dann in Vordergrund wenn man, wie in *Friend Finder*, im Sekundentakt verschlüsselte Positionsdaten versenden möchte.

4.2 Alternative location awareness Verfahren

Neben dem in *Friend Finder* implementierten, recht einfachen Verfahren, um andere Nutzer zu lokalisieren wurden schon andere Verfahren vorgestellt. Im folgenden Abschnitt werden alternative Vorgehensweisen vorgestellt und erklärt.

FriendSensing: Recommending Friends Using Mobile Phones

Im Paper *FriendSensing: Recommending Friends Using Mobile Phones* [Quercia, 2009] aus dem Jahr 2009 erläutern die Autoren ihren Ansatz um Freunde für mobile, soziale Netze zu finden. Hierfür benutzen sie zum einen *Bluetooth* und zeichnen andere Telefone in Reichweite auf. Somit kann aufgezeichnet werden, wie oft Nutzer A und Nutzer B aufeinandergetroffen sind. Des weiteren haben sie eine Software auf den mobilen Geräten installiert, welche aufzeichnet wie oft Nutzer A mit Nutzer B kommunziert. Aus diesen zwei gewonnenen Datensätzen waren die Autoren nun mit Hilfe von verschiedenen Algorithmen, wie zum Beispiel den *Shortest Path* Algorithmus oder *Markov Ketten*, in der Lage zu ermitteln, wie gut Nutzer A einen Nutzer B kennt.

4.3 Zusammenfassung

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 sie durch die hier gewählte Methode nur in eben diesen Shell's gelten wo 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 Packetinformationen 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 und als Zielsystem ein *ARM-Prozessor* gesetzt 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 durchgelofen, so 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

```
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 sollen diese in das gleiche Verzeichniss, wie die vorhergegangenen Archive, entpackt werden. Nun müssen noch die Dateien, welche die Packetinformationen beinhalten für die heruntergeladen Dateien 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/freetyp
```

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 zu dieser Lösung muss nun einfach der ”freetype“-Ordner um eine Ebene nach oben kopieren werden, da die *include*-Pfade in den Headern von ”freetype2“ stellenweiße 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 allererst 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 zuerst das “autogen.sh” Skript ausgeführt.

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

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

Edje

Auch hier gilt wieder, Dateien herunterladen.

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

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

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

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

```
make ; make install
```

Elementary

Zuerst müssen auch hier die benötigten Daten heruntergeladen werden.

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

Nun muss auch wieder das "autogen.sh" Skript heruntergeladen werden.

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

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

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

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

```
make ; make install
```

Weitere Schritte

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

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

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

Anhang 2

Archive für *Eet*:

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

Anhang 3

Archive für *Evas*:

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

Anhang 4

efl_zip.sh:

```
 #!/bin/sh

rm -rf efl/
rm -f efl.zip

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

cp bin/eet.exe efl/
cp bin/libdl-0.dll efl/
cp bin/libevil-0.dll efl/
cp bin/libeina-0.dll efl/
cp bin/libeet-1.dll efl/
cp bin/libevas-0.dll efl/
cp bin/libecore-0.dll efl/
cp bin/libecore_evas-0.dll efl/
cp bin/libecore_job-0.dll efl/
cp bin/libecore_wince-0.dll efl/
cp bin/libembryo-0.dll efl/
cp bin/libedje-0.dll efl/

arm-mingw32ce-strip efl/libdl-0.dll
arm-mingw32ce-strip efl/libevil-0.dll
arm-mingw32ce-strip efl/libeina-0.dll
arm-mingw32ce-strip efl/libeet-1.dll
arm-mingw32ce-strip efl/libevas-0.dll
arm-mingw32ce-strip efl/libecore-0.dll
arm-mingw32ce-strip efl/libecore_evas-0.dll
```

```
arm-mingw32ce-strip efl/libecore_job-0.dll
arm-mingw32ce-strip efl/libecore_wince-0.dll
arm-mingw32ce-strip efl/libembryo-0.dll
arm-mingw32ce-strip efl/libedje-0.dll


cp lib/eina/mp/eina_chained_mempool.dll efl/eina/mp
cp lib/eina/mp/eina_fixed_bitmap.dll efl/eina/mp
cp lib/eina/mp/eina_pass_through.dll efl/eina/mp

arm-mingw32ce-strip efl/eina/mp/eina_chained_mempool.dll
arm-mingw32ce-strip efl/eina/mp/eina_fixed_bitmap.dll
arm-mingw32ce-strip efl/eina/mp/eina_pass_through.dll


cp lib/evas/modules/engines/buffer/mingw32ce-arm/module.dll \
efl/evas/modules/engines/buffer/mingw32ce-arm/engine_buffer.dll

cp lib/evas/modules/engines/software_16/mingw32ce-arm/module.dll \
efl/evas/modules/engines/software_16/mingw32ce-arm/\
engine_software_16.dll

cp lib/evas/modules/engines/software_16_wince/mingw32ce-arm/module.dll \
efl/evas/modules/engines/software_16_wince/mingw32ce-arm/\
engine_software_16_wince.dll

cp lib/evas/modules/engines/software_generic/mingw32ce-arm/module.dll \
efl/evas/modules/engines/software_generic/mingw32ce-arm/\
engine_software_generic.dll

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

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

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

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

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

```
efl/evas/modules/loaders/xpm/mingw32ce-arm/loader_xpm.dll

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

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

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

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

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

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

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

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

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

zip -r -9 efl.zip efl/
```

Literatur

[Android] *Android*. – URL http://www.android.com/. – [Online; letzter Aufruf 25.01.2010]

[CeGCC] *CeGCC*. – URL http://cegcc.sourceforge.net/. – [Online; letzter Aufruf 20.11.2009]

[efl] *Enlightenment*. – URL http://www.enlightenment.org/. – [Online; letzter Aufruf 20.11.2009]

[iPhoneOS] *iPhone OS*. – URL http://www.apple.com/de/iphone/. – [Online; letzter Aufruf 03.02.2010]

[IRC] *IRC-Protokoll*. – URL http://www.irc.org/. – [Online; letzter Aufruf 27.01.2010]

[Kerberos] *Kerberos*. – URL http://www.kerberos.org/. – [Online; letzter Aufruf 28.01.2010]

[OpenSSL] *libCrypto*. – URL http://www.openssl.org/. – [Online; letzter Aufruf 25.01.2010]

[libircclient] *libircclient*. – URL http://libircclient.sourceforge.net/. – [Online; letzter Aufruf 25.01.2010]

[openPGP] *OpenPGP*. – URL http://www.openpgp.org/. – [Online; letzter Aufruf 28.01.2010]

[PalmOS] *PalmOS*. – URL http://www.palm.com/. – [Online; letzter Aufruf 25.01.2010]

[PGPmobile] *PGP Mobile*. – URL http://www.pgp.com/products/mobile/index.html. – [Online; letzter Aufruf 28.01.2010]

[SymbianOS] *Symbian OS*. – [Online; letzter Aufruf 03.02.2010]

[WebOS] *WebOS*. – URL http://palmwebos.org/. – [Online; letzter Aufruf 25.01.2010]

[Windows] *Windows Mobile*. – URL http://www.microsoft.com/windowsmobile/de-de/default.mspx. – [Online; letzter Aufruf 25.01.2010]

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

[Klaus Rechert 2009] KLAUS RECHERT, Konstantin W.: *Spontaneous Privacy-Aware Location Sharing*. 2009

[Landon P. Cox 2007] LANDON P. COX, Varun M.: *SmokeScreen: Flexible Privacy Controls for Presence-Sharing*. 2007

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