Ausarbeitungen zum Seminar

Handy-Betriebssysteme

Wintersemester 2012/13

18. Februar 2013

Fakultät II – Informatik, Wirtschafts- und
Rechtswissenschaften
Department für Informatik
Abt. Systemsoftware und verteilte Systeme

www.svs.informatik.uni-oldenburg.de

Inhaltsverzeichnis

Die historische Entwicklung der Telefonie von den Anfängen bis heute

Hendrik Grunau
Universität Oldenburg
Deutschland
hendrik@grunau.info

***Zusammenfassung*—In dieser Arbeit wird die Entwicklung der Telefonie von der ersten Technologie bis zur heutigen Mobilfunktechnologie beleuchtet.**

I. Ursprung des Telefons

A. Die Anfänge der Technologie

Die erste Stimmübertragung wurde 1861 durch den Physiklehrer Phillip Reis ermöglicht. Hierzu verwendete er eine Schweinsblase, welche einen Hebel zu einem Kontakt bewegen konnte. Dadurch wurden akustische Schwingungen in einen Strom umgesetzt. Zur Rückwandlung war am anderen Ende des Drahtes eine Stricknadel befestigt, welche den Boden einer Zigarrenkiste zum hörbaren Schwingen brachte. Mit diesem Fernhörer waren gleichzeitig auch der erste Lautsprecher und das erste Kontaktmikrofon entwickelt. Der erste Übertragene Satz mit diesem Telefon lautete: „Das Pferd frisst keinen Gurkensalat.“, da man hier fehlende Worte nicht aus dem Rest des Satzes ableiten konnte. Jedoch war dieser Apparat durch Erschütterungen und Temperaturänderungen leicht zu stören, wodurch er nur selten funktionierte und nicht alltagstauglich war.

Graham Bell stellte 1874 eine Weiterentwicklung des Fernhörers vor, welcher nun aus einer Metallmembran bestand, die direkt durch einen elektrischen Strom und magnetische Induktion beeinflusst wurde. Zwei Jahre später, 1876, ließ er sein verbessertes Telefon patentieren, welches nun auf Lautsprecher und Mikrofon aus Induktionsspulen basierte, und schaffte damit den Durchbruch zur praktischen Relevanz der Erfindung. Bell verwendete für seine Erfindung einen Stabmagneten, der die Schwingungen einer Metallmembran über eine Spule in Wechselstrom umgewandelte. Dieses Prinzip war auch umzukehren, sodass über die Spule durch Wechselstrom hervorgerufene Induktion die Metallmembran zu Schwingunen anregte, und so Töne erzeugte. Durch Verwendung eines Hufeisenmagneten statt eines Stabmagneten konnten größere Lautstärken erzeugt werden. Noch im selben Jahr wurde hiermit das erste Ferngespräch über eine zwei Meilen lange Telegrafenleitung von Cambridge nach Boston durchgeführt.

Der Weltweite Durchbruch des Bellschen Telefons gelang 1877, nachdem Benjamin Osgood Peirce, ein Freund und Mitarbeiter Bells, das Gerät zu einer handlichen Form (vgl. Abb. 1) verkleinerte und im „Scientific American“ vom

Abbildung 1. Telefonmodell nach Peirce [1]

sechsten Oktober vorstellte. Mit diesem Modell begann die Geschichte des Fernsprechens am 26.10.1877 in Deutschland.

Bells Konstruktion des Fernsprechers konnte aufgrund der schwachen erzeugten Ströme Schall nur über relativ geringe Distanzen übertragen. Deswegen versuchte Werner von Siemens die eintreffenden Schwingungen zu verstärken. Dies gelang ihm durch zwei Veränderungen des Ursprungsdesigns von Bell. Zum einen verbaute auch er einen Hufeisenmagneten statt des Stabmagneten. Zusätzlich dazu wurde die schwingenden Eisenplatte durch eine trichterförmigen Membran ersetzt, an der eine leichte Drahtrolle zum schwebenden Schwingen in einem starken Magnetfeld befestigt war. So wurden in der Drahtrolle kräftige Wechselströme induziert. Siemens stelle seine Entwicklung 1878 als Elektrodynamisches Telefon vor.

Für die nächsten neun Jahre war das Mikrophon meist fest, aber höhenverstellbar in den Wandapparat mit Einsprechtrichter installiert. Lautsprecher waren an einer Schnur befestigt, damit sie an das Ohr gehalten werden konnten. Da die Lautstärke bei Geräten der Anfangszeit noch gering war, waren einige Geräte mit zwei Fernhörern ausgestattet, welche dann an beide Ohren gehalten wurden. Mit den laufenden Verbesserungen der Gesprächsqualität wurde der zweite Hörer weggelassen, oder für einen weiteren Zuhörer verwendet. 1887 wurde dann von der Firma Mix und Genest ein neuartiger Fernhörer mit integriertem Mikrofon, der Telefonhörer, der Öffentlichkeit vorgestellt.

B. Die ersten Fernsprechnetze

Die ersten Fernsprechnetze wurden in Deutschland ab 1881 installiert. So begann das Projekt am zweiten Januar mit einer Versuchsanlage in Berlin, welche acht Teilnehmer

umfasste. Vier Monate später wurden bereits 48 vermittelt, darunter neun Börsensprechstellen. Weitere vier Monate später erschien das erste Telefonbuch für Berlin. Es umfasste neben 94 Netzteilnehmern auch eine Anleitung des Fernsprechers.

Die Kosten eines einfachen Telefonanschlusses bis zu einer Länge von zwei Kilometern betrugen 200 Reichsmark, was für damalige Verhältnisse sehr viel war. Für jeden Kilometer über dieser Strecke wurden weitere 50 Reichsmark veranschlagt. Dadurch war das Telefon ein Statussymbol der Wohlhabenden. Nach einem ähnlichen Verkaufsmodell wurden die Fernsprechgebühren berechnet. Für alle Gebühren im Orts- und Ferndienst wurde ein Pauschaltarif verlangt. Für Gespräche in der Nacht oder außerhalb der normalen Dienstzeiten der Vermittlungsstellen wurden Zuschläge auf den Tarif erhoben.

C. Das erste Wählverfahren

Bisher war nur die Verbindung von genau zwei Endgeräten möglich. Deswegen mussten in den Vermittlungszentralen der ersten Netze die Verbindungen manuell durch Angestellte gesteckt werden. Für kurze Zeit waren nur Männer in Vermittlungszentralen tätig. Diese wurden aber schnell durch Frauen ersetzt, da die höhere Stimmfrequenz bei schlechterer Leitungsqualität besser verständlich war. Damit war das „Fräulein vom Amt“ geboren.

Bis zur Automatisierung des Wählverfahrens wurden in Vermittlungsstellen Fernsprechhandvermittlungseinrichtungen, besser bekannt als Klappenschränke, verwendet. Für jeden Netzteilnehmer war hier eine Klinke und eine Fallklappe vorgesehen. Diese Klappe wurde durch einen Elektromagneten ausgelöst. Bei einem Gesprächswunsch wurde an den Fernsprechapparaten der Zeit über einen Kurbelinduktor ein Wechselstrom erzeugt, welcher den entsprechenden Elektromagneten des Schrankes anregte, wodurch die zugehörige Klappe herunterfiel und so den Verbindungswunsch anzeigte. Nun verband das Fräulein vom Amt ihren Sprechapparat mit der Klinkenbuchse des Teilnehmers, um zu erfahren mit wem dieser Verbunden werden wollte. Das Ende des Gesprächs wurde durch erneutes Erregen des Elektromagneten über den Kurbelinduktor durch den anrufenden Teilnehmer angezeigt. Beendet wurde die Verbindung dann über die Vermittlungskraft, welche die Klinkenstecker entfernte und die Klappe manuell wieder in Ausgangsposition brachte, damit der nächste Verbindungswunsch erfasst werden konnte.

II. Netzentwicklung

A. Die Anfänge des analogen Netzes

Im Jahr 1883 wurde das erste Fernamt Berlins in Betrieb genommen. Zu diesem Zeitpunkt waren 1652 Anschlüsse mit 35 Handvermittlungsstellen verzeichnet. Von hier wurde noch im selben Jahr über eine Strecke von 150 Kilometer die erste Fernleitung nach Magdeburg gelegt. Dieses Kabel war eine Doppelleitung, welche erstmals Pupinspulen zu Verbesserung des Dämpfungsverlaufs besaß. In den darauf folgenden zwölf Jahre wurden Leitungen zwischen den größeren deutschen Städten gelegt, wodurch einige Ortsnetze nun miteinander verbunden waren. Zusätzlich existierte ab August 1900 die erste Fernverbindung zwischen Berlin und Paris. Das erste Untersee-Fernsprechbabel wurde 1906 im Bodensee verlegt. Dieses 12,6 Kilometer lange, mit einer Papierisolation und einem Bleimantel versehene Kabel war mit 22 Pupinspulen bestückt, welche sieben Aderpaare ermöglichten.

Bis zum Jahr 1912 wurden in Deutschland die Fernverbindungen fast ausschließlich über Oberflächenleitungen hergestellt. Diese Freileitungen wurden in Wintermonaten durch Eis und Schnee oft beschädigt, teilweise wurden die Masten durch Stürme umgeknickt. Deswegen wurde Mitte des Jahres 1912 der Auftrag für ein unterirdisches Fernkabel, das Rheinlandkabel, an die Firma „Siemens & Halske“ vergeben, welche bereits das Unterseekabel verlegt hatte. Im ersten Bauabschnitt 1913/1914 wurde begonnen auf der Strecke von Berlin nach Hannover ein mit Pupinspulen versehenes, verdrilltes Kabel zu legen. Da die 1912 erfundenen Leitungsverstärker noch nicht genügend erprobt waren, war diese Fernleitung für eine verstärkungslose Verbindung bis ins Rheinland konstruiert. Die Bauarbeiten wurden 1914 durch den ersten Weltkrieg unterbrochen. Dennoch wurden im gleichen Jahr durch steigend Nachfrage und mangelnde Kabelressourcen Untersuchungen angestellt um mehrere Kanäle im Trägerfrequenzverfahren über eine Leitung zu übertragen. Auch wurden erstmals Telegrafie und Telefongespräche zeitgleich auf der selben Leitung übertragen. Dieses Prinzip der Unterlagerung wurde in den folgenden Jahren zur wirtschaftlichen Nutzung der Kabel verstärkt ausgebaut.

Die erste Übertragung im Trägerfrequenzverfahren über 100 Kilometer Freileitung gelang nach Erfindung der Elektronenröhre im Jahr 1918. Zwei Jahre später entstanden die ersten Verstärkerämter. Durch die während des Krieges erfundene Gabelschaltung konnte auf Doppelleitungen die Sprache richtungsgetrennt verstärkt werden. Voraussetzung war jedoch eine Verringerung der Bespulung der Kabel. Da die Pupinspulen in Spulenkästen längs der Kabelstraßen untergebracht waren, stellte dies allerdings kein großes Problem dar. 1932 gelang es schließlich das Trägerfrequenzverfahren auf Fernkabeln einzusetzen. Hierzu musste die Bespulung der Kabel vollständig entfernt werden.

B. Die Automatisierung des analogen Netzes

Das erste Patent für eine automatische Vermittlungstechologie wurde bereits 1879 in den USA vergeben. Die erste technisch durchführbare Lösung eines automatisierten Wählverfahrens folgte allerdings erst zwölf Jahre später, 1891, durch Almon Strowger. Sein Patentantrag für den weltweit erfolgreichen Strowger Wähler wird heute als die Geburtsstunde der automatisierten Vermittlungstechnik bezeichnet. Das Funktionsprinzip des Strowger Wählers

orientierte sich stark an der Arbeitsweise der Handvermittlungsschränke. Es waren mehrere Reihen von je zehn halbkreisförmig angeordneten Schleifkontakten, die zu den entsprechenden Teilnehmerleitungen gehörten, untereinander angebracht. So konnten durch einen Wähler 100 Ausgänge vermittelt werden. Im Gegensatz zur Handvermittlung wurde bei Verbindungswunsch kein Klinkenkabel verwendet, sondern ein beweglicher Kontaktarm, welcher sich vertikal und horizontal bewegen konnte, um alle Kontakte zu erreichen. Die Steuerung eines Hebdrehwählers erfolgte durch elektrische Impulse, welche beim Strowger Wähler jeweils einen von zwei Elektromagneten erregten. So konnten nacheinander zwei Koordinaten für eine Heb- und eine Drehbewegung verarbeitet werden. Ein dritter Magnet stellte nach Gesprächsende die Ausgangslage des Kontaktarms, wieder her. Bis zur Erfindung der Wählscheibe wurde ein von Almon Strowger erfundenes Vermittlungssystem verwendet. Dazu besaß das Telefon für jede zu wählende Stelle der Nummer eine eigene Taste, welche entsprechend des Wertes der Stelle gedrückt werden musste. Für die Nummer „042“ folgte so ein viermaliges drücken der 10er-Taste und ein doppeltes drücken der 1er-Taste. Die vorhandene 100er-Taste wurde in diesem Fall ausgelassen. Mit dem System folgte, dass neben der für das Gespräch benötigten Doppelader für jede dieser Wähltasten ein weiteres Verbindungskabel zur Vermittlungsstelle vorhanden sein musste. Ab 1898, mit Anerkennung des Patentes von Keith und Erickson auf die Wählscheibe, wurden die Kontaktunterbrechungen direkt durch die gewählten Nummern erzeugt. Dadurch war es wieder möglich die bereits vorhandenen Doppeladerkabel zu verwenden. Benutzt wurde die Fingerlochscheibe, indem mit dem Finger das Loch der gewünschten Ziffer ausgefüllt, und dann bis zum Anschlag gedreht wurde. Loslassen führte zu einer Rückdrehung in Ausgangsposition. War die Nummer vollständig eingegeben, musste dies dem Wählsystem signalisiert werden, indem ein weiterer Knopf gedrückt wurde.

Die erste automatische Versuchszentrale wurde 1892 in La Porte (Indiana) mit 80 Anschlüssen installiert, von denen 55 genutzt wurden. Jedoch war hier das Telefongeheimnis nicht gewährleistet, da Teilnehmer sich direkt auf eine Verbindung aufschaltete und so in laufende Gespräche integriert wurde. In Europa wurde die erste automatisierte Vermittlungszentrale 1908 in Hildesheim mit 900 Teilnehmern in Betrieb genommen. Die hier verwendeten Strowger Wähler waren in verschiedene Kategorien eingeteilt: Gruppenwähler und Leitungswähler. Die letzten beiden Ziffern der Rufnummer waren für den Leitungswähler bestimmt, welcher damit den gewünschten Anschluss wie oben beschrieben verband. Pro Gruppenwähler wurde eine weitere Ziffer in der Rufnummer benötigt, welche für diesen die vertikale Ausrichtung und damit die Untergruppe festlegte. In der horizontalen wurde selbstständig durch den Wähler eine freie Verbindung herausgesucht. Weitere Funktionen wie Gebührenzählung einleiten oder Freizeichen senden waren auf den Leitungswähler als letztes Glied der Kette abgebildet. Dieses Freiwahlverfahren stellte einen bedeutenden Fortschritt für die Anfänge der Informatik dar, da hier die Verbindungsmöglichkeit vor der Herstellung zuerst geprüft wurde.

Die Implementierung des automatisierten Verfahrens war nur über den Zwischenschritt einer halbautomatischen Vermittlung möglich, welche 1914 in Dresden für rund 14000 Teilnehmer eingerichtet wurde. Hier wurden Anrufer zuerst mit einer Vermittlungskraft verbunden, welche dann die Teilnehmernummer tippte und automatisch verbinden ließ, da die Anzahl der benötigten Wähltelefone nicht bereitgestellt werden konnte. Der nächste Entwicklungsschritt wurde durch den ersten Weltkrieg verzögert. 1923 wurde dann in Weilheim die erste automatische Fernvermittlungsstelle mit mehreren Knotenpunkten in Betrieb genommen. Da hier alle Teilnehmer zuerst über eine teure Fernleitung mit der Zentralstelle verbunden wurden, wurde später ein Umsteuerwandler in die Wähltelefone eingebaut, welcher die Ortskennung der ersten Ziffern der Nummer parallel zur Zentralstelle abglich und bei Übereinstimmung direkt mit der Vermittlungsstelle vor Ort verband, wobei die Fernleitung bereits nach Sekunden wieder freigegeben wurde. 1925 wurde in Berlin das letzte große handvermittelte Fernamt eröffnet, stillgelegt wurde die letzte Handvermittlung allerdings erst im Jahr 1966.

C. Das Digitale Netz

Mit fortschreitender Computertechnologie in den 1960er Jahren wuchs das Bedürfnis Daten auf direkterem Wege als bisher, Speichermedien per Post, zu verschicken. Da das Telefonnetz bereits fast überall verfügbar war, wurde dieses als Übertragungsmedium gewählt. Für die Datenfernübertragung (DFÜ) über das Telefonnetz wurde schon 1960 das erste Modem, abkürzend für *modulation* und *demodulation*, von den Bell Telephone Laboratories mit einer Übertragungsrate von etwa 300 Bit/s vorgestellt. In Deutschland wurde das erste Modem erst 1966 durch die Deutsche Bundespost entwickelt, wessen Übertragungsrate mit 1200 Bit/s allerdings vier mal so groß wie die des Bell-Modells war.

Abbildung 2. Akustikkoppler-Modem [2]

Zur Datenübertragung wurde bei den ersten Modems ein Telefonhörer auf entsprechende Kontakte des Akustikkopplers (vgl. Abb. 2) gelegt, damit akustische Signale aus dem Telefonnetz empfangen und dorthin versendet werden konnten. Die Wandlung von und zu digitalen Signalen erfolgte im Modem über einfache Amplituden- und Phasenmodulation. 1969 wird heute als die Geburtsstunde des Internets bezeichnet, was damals das DARPANET, ein von Militär und einigen amerikanischen Universitäten entwickeltes Kommunikationsnetz zwischen den beteiligten Universitäten, war. Als erste Kommunikationsprotokolle folgten zwei Jahre später Telnet und FTP, weitere drei Jahre später wurde TCP veröffentlicht, in dessen Spezifikationen erstmals der Begriff „Internet“ verwendet wurde.

Der nächste große Entwicklungsschritt begann 1979 mit dem Bau der ersten Glasfaserleitung und der Entscheidung das komplette Fernsprechnetz zu digitalisieren, womit vor allem die Vermittlungen gemeint waren. Drei Jahre später wurde entschieden ein Dienste-integrierendes digitales Netzwerk zu errichten. Die Spezifikationen für ISDN (Integrated Services Digital Network) folgten 1984 durch die internationale Telekommunikationsbehörde. Dabei stützte man sich auf das EWSD-System von Siemens, welches bereits 1980 erste digitale Vermittlungen ermöglichte. Zeitgleich wurden die ersten Heimcomputer immer beliebter, wodurch die Datenfernübertragung einer breiten Masse verfügbar war.

Die ersten Pilotprojekte für ISDN entstanden 1987 in Mannheim und Stuttgart, worauf zwei Jahre später der offizielle Praxisdienst folgte. In weiteren fünf Jahren gelang es alle notwendigen Softwareänderungen in den Vermittlungstellen abzuschließen, wodurch ISDN kommerziell verfügbar wurde. Nach einem weiteren Jahr war die flächendeckende Digitalisierung Deutschlands abgeschlossen, wobei die noch nicht digitalisierten Vermittlungsstellen per Fremdaufschaltung umgangen. wurden. Ende 1997 war die Umstellung vollendet. Mit Digitalisierung der Vermittlungen war es nun auch möglich, ein Modem direkt an die Telefonleitung anzuschließen, ohne den Umweg über eine Akustikkopplung gehen zu müssen, wenn man sich mit dem Internet verbinden wollte. Dadurch verringerte sich die Einwahlgeschwindigkeit von etwa 30 auf rund zwei Sekunden.

Für Privatkunden war 1998 mit den Modellen „V.90“ und „V.92“, welche besser als „56K-Modem“ bekannt sind, der Höhepunkt der Modementwicklung erreicht. Hiermit ließ sich unter der Vorraussetzung, dass der Anschluss mit einer digitalen Vermittlungsstelle verbunden war, eine Übertragungsrate von 56kbit/s erreichen. Die Leitungsqualität erlaubte allerdings meistens nur Geschwindigkeiten bis 40kbit/s. Die maximale Geschwindigkeit eines ISDN-Basiskanals (B-Kanal) betrug 64kbit/s, ein Basisanschluss beinhaltete zwei Basiskanäle sowie einen Steuerkanal (D-Kanal) mit 16kbit/s. Für eine Datenübertragung konnten nur B-Kanäle genutzt werden, diese allerdings auch zusammengeschaltet. Für Heimanwender war eine Kaskadierung von sechs Kanälen möglich, also eine Übertragungsrate von maximal 384kbit/s. Für Firmenkunden bestand die Möglichkeit einen Primärmultiplex-Anschluss zu installieren, welcher eine Summenbitrate von 2Mbit/s erreichte, indem 32 B-Kanäle kombiniert wurden.

Mit der rasanten Verbreitung des neuen WWW 1993 wuchs auch der Bedarf an Geschwindigkeit. Mitte der 90er Jahre wurde dann ADSL als Nachfolger von HDSL beziehungsweise SHDSL entwickelt, welches die Grundlage für den verstärkten und dringend benötigten Netzausbau bildete. Durchsetzen konnte sich DSL für Privatanwender allerdings erst mit der Jahrtausendwende, als erste Flatrates verfügbar wurden und massiver Wettbewerb für fallende Preise sorgte. Heutzutage werden neben traditionelleren ISDN-Anschlüssen auch komplette DSL-Anschlüsse angeboten. Telefonieren wird dadurch mehr und mehr mit Voice over IP, dem Telefonieren über das Internet, ersetzt, was den Wandel des Telefonnetzes zum Internet verdeutlicht.

III. Mobiltelefonie

A. Das A-Netz

Die Geschichte der mobilen Telefonie begann 1952 in Bremen. Hier wurde das erste Dokumentierte Autotelefon mit 16 Kilo Gewicht und einem Kostenpunkt von drei VW Käfern in ein Taxi installiert. Bis 1958 bestand in Deutschland noch kein einheitliches Mobilfunknetz. Stattdessen existierten viele unzusammenhängende Hafen-, Zug-, und Stadtfunknetze, welche noch nicht für zivile Nutzung vorgesehen waren. Dies änderte sich mit der Einführung des A-Netzes, bei der diese in ein öffentliches Netz, dem weltweit größten seiner Art, unter der Vorwahl 010 Zusammengefasst wurden. Technisch bedingt war die Kapazität auf maximal 11 000 Teilnehmer beschränkt, welche sich aufgrund hoher Anschaffungs- und Haltungskosten aus wenigen Großunternehmern und Politikern zusammensetzten. Die Gespräche des etwa 80% deckenden Netzes wurden von insgesamt 60 Vermittlungskräfte verbunden, sofern der Aufenthaltsort des gewünschte Teilnehmers in Deutschland war. Problematisch hierbei war den Funkbereich einer Zelle zu verlassen. Da die Gespräche über die Landesvermittlung gehalten wurden, führte ein Bereichswechsel zum Verbindungsverlust mit dieser, wodurch das Gespräch zusammenbrach.

B. Das B-Netz

14 Jahre später, 1972, wurde mit dem B-Netz der offizielle Nachfolger des A-Netzes eingeführt und über sechs Jahre flächendeckend ausgebaut. Damit war auch im Mobilnetz der Selbstwählbetrieb eingeführt. Hierfür war das Sendegebiet in nummerierte Zonen unterteilt, welche zusammen mit der Ortskennzahl Vorwahlnummern bildeten. So besaß jedes Gebiet eine eigene Vorwahl, welche dem Anrufer für einen Verbindungsaufbau bekannt sein mussten. Zusätzlich dazu konnten durch begrenztes Roaming erstmals Mobilgespräche mit Österreich, Luxemburg und den Niederlanden

getätigt werden, die Funkzelle konnte allerdings noch nicht ohne Gesprächsunterbrechung gewechselt werden. Wegen Größe und Gewicht wurden weiterhin meist Autotelefone verwendet, welche die für das Telefongeheimnis nötige Verschlüsselung nur mit zusätzlicher, sehr teurer Technologie, gewährleisteten. Dadurch wurde ein Großteil der Gespräche unverschlüsselt übertragen und konnte durch Funkanlagen abgehört werden. Trotz hoher Kosten war das Netz bereits Ende des Jahrzehnts ausgelastet, sodass die Frequenzen des 1977 deaktivierten A-Netzes integriert werden mussten. In Spitzenzeiten umfasste das B-Netz etwa 27 tausend Teilnehmer, bis es 1994 abgeschaltet wurde.

C. Das C-Netz

Mit der flächendeckenden Einführung des C-Netzes 1985 war der nächste große Entwicklungsschritt getan. Durch den Einsatz des Home Location Register-Verfahrens (HLR) war es nun möglich Anrufe ohne Wissen um die genaue Position des Gesprächspartners zu tätigen, da das System diese selbstständig erfasste. Auch wurde ein Hand Over Verfahren implementiert, welches den Wechsel der Funkzelle ohne Gesprächsabbruch ermöglichte. Allerdings betrieben die Nachbarländer, anders als beim B-Netz, das C-Netz mit einem anderen Standard, was Roaming effektiv verhinderte und somit einer der Hauptgründe für die Weiterentwicklung der Netztechnologie war.

Eine weitere Neuerung des C-Netzes war das Lösen der Bindung von Telefonnummer und Gerät, wie sie im A- und B-Netz zu finden war. Stattdessen wurde sie auf einem Magnetstreifen einer Berechtigungskarte, dem Vorläufer der SIM-Karte, gespeichert. Weiterhin waren nun alle Netzteilnehmer bundesweit wieder mit einer einheitlichen Vorwahl, 0161, verknüpft. Der Haupteinsatzbereich des auf etwa 850 tausend Teilnehmer gewachsenen C-Netzes verschob sich nur wenig im Vergleich zu den vorherigen Netzen, neben Autotelefonen waren nun auch Küstenschifffahrt und Eisenbahntelefone enthalten. Dennoch wurden die ersten standardisierten portablen Telefone eingeführt. Sie besaßen ein Gewicht von etwa fünf Kilogramm und konnten ohne fest installierte Sendestation betrieben werden, was sie zu den ersten wirklichen Mobiltelefonen machte. Ebenfalls wurde die Sicherheit der Verbindungen durch Einführen einer Verschleierungsfunktion verbessert, welche das Mithören von Gesprächen erschwerte.

D. Die digitalen Funknetze D und E

Im Sommer 1992 startete die Deutsche Telekom mit dem D1-Netz das erste digitale Funknetz. Wenig später wurde von Mannesmann Mobilfunk allerdings ein zweites, ebenfalls auf dem GSM-9000-Standard basierendes Funknetz, das D2-Netz, in Betrieb genommen, wodurch es zur ersten Konkurrenzsituation der Mobilfunkgeschichte kam. Roaming, welches nun in ganz Europa verfügbar war, konnte aufgrund abweichender Technologien allerdings erst ab 1995 durch Zusatzfunktionen der D-Netze auch in den USA genutzt werden. Durch die Digitalisierung konnten nun auch andere Datentypen übertragen werden. Davon machte ab 1994 das Short-Message-System gebrauch, welches ursprünglich dazu gedacht war den Netzteilnehmer mit kleinen Informationen vom Anbieter zu versorgen. Mit fortschreitender Technologie wurden die Endgeräte immer weiter verkleinert und begannen den Weg zu Smartphones. Zu Einführung des neuen Netzstandards lagen die Preise für Mobiltelefone noch bei etwa drei tausend Euro. Mit steigender Nachfrage und neuerlicher Konkurrenz durch das E-Netz sanken diese aber schnell.

Mit diesem startete 1993 E-Plus als dritter Anbieter, vorerst auf den Berliner Raum konzentriert, ein Mobilfunknetz. Anders als die D-Netze war es speziell auf energiesparende Handys ausgelegt, welche durch geringere Sendeleistung ein engmaschigeres Netz von Sendemasten benötigten. Weiterhin verwendete das E-Netz Frequenzen im Bereich von 900 MHz, wohingegen die D-Netze das Frequenzband um 1800 MHz nutzten. Dadurch war es nicht möglich, ein Handy in beiden Netzarten zu betreiben. Dies änderte sich erst Ende der 90er Jahre mit Einführung der Dual- und Tri-Band-Telefone. Mit der Zuweisung von D- und E-Netz Frequenz an das jeweils andere Funknetz zur Kapazitätserhöhung durch die Bundesnetzagentur im Jahre 2006 lässt sich die Unterscheidung anhand der genutzten Frequenzbänder nicht mehr durchführen, sodass die Netze mehr oder weniger zu einem verwoben sind. Insgesamt konnten die Netze im zweiten Quartal 2011 etwa 110 Millionen Teilnehmer verzeichnen.

2004 wurde der GSM-Standard im digitalen Netz langsam durch UMTS ersetzt. Damit wurde die mit GSM eingeführte paketorientierte Datenübertragung um das Internet-Protokoll (IP) ergänzt, welches allerdings nicht für Gespräche genutzt wurde. Dadurch wurde die Datenübertragungsrate von etwa 14kbit/s auf maximal 220kbit/s gesteigert. Zusätzlich waren nun neue Funktionen wie ein Konferenzanruf an mehrere Teilnehmer oder gleichzeitiges telefonieren und surfen möglich. Durch den Dualmode waren alle UMTS-Endgeräte zu GSM abwärtskompatibel. Mit HSPA und HSPA+ als Softwareerweiterungen wurde die Geschwindigkeit auf bis zu 21 Mbit/s gesteigert.

Im Jahr 2010 wurde LTE als UMTS-Nachfolger eingeführt, welches nun auch Telefonie mit IP realisiert und Datenraten bis 100Mbit/s unterstützt. Dabei wurde es ergänzend installiert, da die bestehenden Netze nur nachgerüstet werden mussten und ein paralleler Betrieb mit UMTS möglich ist.

Die Erweiterung des LTE-Standards, dass noch nicht verfügbare LTE Advanced, kann mit bis zu 1Gbit/s übertragen. Für diese Umrüstung ist neben neuen Endgeräten ebenfalls nur ein Softwareupdate der Sendestationen vonnöten. Erste kommerzielle Versionen der Technologie sollen 2013 verfügbar sein.

LITERATUR

[1] (cc) by-nc-sa, Lehrstuhl für Nachrichtentechnik, RWTH Aachen

[2] (pd) Lorax on en.wikipedia

[3] http://www.bayern-online.com/v2261/artikelliste.cfm/203/Geschichte-Telekommunikation-bis-1999.html

[4] http://www.auslandsvorwahl.info/historie_telefon.php

[5] http://www.fernmeldemuseum-aachen.de/html/zeitFon.html

[6] http://www.uni-protokolle.de/Lexikon/Geschichte_des_Telefons.html#Die_%E2%80%9EErfindung%E2%80%9C_des_Telefons

[7] http://www.vdsl-tarifvergleich.de/vdsl-technik/geschichte-von-isdn-ueber-dsl-bis-vdsl.html

[8] http://history-computer.com/ModernComputer/Relays/Stibitz.html

[9] http://www.computerwoche.de/a/isdn-schnelle-zufahrt-auf-die-datenautobahn,1089013

[10] http://www.wissen.de/die-geschichte-der-mobiltelefone

[11] http://www.focus.de/digital/handy/handygeschichte_did_12098.html

[12] http://www.mobilfunk-geschichte.de

[13] http://www.ighft.de/

[14] www.telefongeschichte-owl.de/

[15] Günther Mergelsberg: Das Telefon und seine Entwicklung. 2 Bände, Sammler- und Interessengemeinschaft für das historische Fernmeldewesen e. V., Bad Homburg 1996.

[16] http://www.elektronik-kompendium.de/sites/kom/index.htm

[17] http://www.telespiegel.de/html/handy.html

[18] http://www.lte-anbieter.info/lte-geschichte.php

[19] http://www.lte-advanced-verfuegbarkeit.com/

[20] http://www.golem.de/1106/84529.html

Prozessorarchitekturen moderner Handys (insb. ARM)

Stephan Wessels
Department für Informatik
Carl von Ossietzky Universität Oldenburg

***Zusammenfassung*—Moderne Handys dienen mittlerweile als Allzweck-Computer. Sie stellen hohe Anforderungen an ihren Prozessor und dessen Leistungsfähigkeit. Im Gegensatz dazu steht die Forderung nach einem geringen Energie- und Platzbedarf, sowie geringen Kosten. Das RISC-Design bietet einen Ansatz um diesen Problemen entgegen zu kommen. Ein simpler Befehlssatz mit einheitlichen und schnellen Instruktionen liefert eine effiziente Rechenleistung. Die ARM-Prozessorarchitektur nimmt sich genau diesen Ansatz zum Vorbild. Es soll gezeigt werden, wie die 32-Bit ARMv7-Architektur aufgebaut ist. Dabei wird der große Registersatz vorgestellt und auf den simplen Befehlssatz eingegangen. Genauer wird die Kodierung der Datenverarbeitungsbefehle besprochen. Weiterhin wird die Möglichkeit durch Thumb-Befehle einen dichteren Code zu erzeugen und das big.LITTLE-Konzept, zur Steigerung der Energieeffizienz, angerissen. Abschließend folgt ein Ausblick auf den kürzlich angekündigten Nachfolger ARMv8, welcher zum ersten Mal 64-Bit unterstüzt.**

I. Einleitung

Handys sind aus unserem Leben nicht mehr wegzudenken. Der Großteil der Menschen in Deutschland besitzt ein mobiles Telephon und benutzt dieses regelmäßig. Dabei schreitet die Entwicklung rasant voran und in den letzten Jahren haben die Smartphones einen regelrechten Boom erlebt. Diese übernehmen mittlerweile viele Funktionalitäten eines modernen Desktop-PCs.

Ein Hauptmerkmal der Geräte bleibt aber weiterhin die Mobilität, die durch geringe Bauform und fortschrittliche Akkutechnik gegeben ist. Damit trotz steigender Leistung die Geräte mobil und handlich bleiben, werden an die Hardware spezielle Anforderungen gestellt. So ist z.B. der Prozessor ein wichtiger Faktor im Hinblick der Effizienz. Im Folgenden wird darauf eingegangen, welche Bedingungen Handy-Prozessoren erfüllen müssen und welche Architektur speziell auf diese Anforderungen zugeschnitten ist.

II. Anforderungen an Handy Prozessoren

Handys sind mobile Geräte. Dementsprechend werden an ihre Prozessoren spezielle Anforderungen gestellt, die sich von einem Desktop-PC Prozessor unterscheiden. Besonders die folgenden Faktoren spielen eine Rolle:

- **Stromverbrauch** - Handys sollen mobil sein und müssen durch Akkus versorgt werden. Diese bieten aber nur eine begrenzte Kapazität. Um trotzdem die Laufzeit des Gerätes zu maximieren muss ein stromsparender Prozessor verbaut werden.
- **Wärmeentwicklung/Platzbedarf** - Handys müssen handlich sein. Es wird dabei viel Wert auf ein kleines und flaches Gehäuse gelegt. Durch den geringen Platz ist es nicht möglich einen Lüfter einzubauen, weshalb sich die Abwärme der elektronischen Bauteile im Gehäuse staut. Daher darf der Prozessor nicht übermäßig viel Wärme produzieren und sollte zudem so klein wie möglich sein.
- **Kosten** - Handys sind oft auf ein breites Käuferspektrum zugeschnitten, für die der Preis des Gerätes entscheidend ist. Um die Verkaufskosten dementsprechend niedrig zu halten, darf auch der Prozessor nicht viel kosten.
- **Leistung** - Moderne Handys stellen eine Vielzahl an Funktionen bereit. Diese sollen für den Benutzer schnell und teilweise auch zur gleichen Zeit verfügbar sein. Somit werden hohe Anforderungen an die Rechenleistung des Prozessors gestellt.

Da mobile Telephone in der Vergangenheit überwiegend zur Kommunikation benutzt wurden stand der Leistungsaspekt im Hintergrund. Mit der Zeit jedoch wurden immer mehr Funktionen benötigt um auf dem Markt überzeugen zu können. Das hat dazu geführt, dass Handys heute nicht mehr nur an ihrer Akkulaufzeit, sondern auch an der Leistung des Prozessors gemessen werden. Die ersten drei Anforderungen stehen im Gegensatz zur Letzten. Wird die Leistung erhöht, so hat dies z.B. oftmals eine gesteigerte Wärmeentwicklung zur Folge. Daraufhin stellt sich die Frage, welche Prozessorarchitektur einen guten Kompromiss aus geringem Stromverbrauch, Platzbedarf und Kosten, sowie trotzdem hoher Leistung bietet.

III. RISC

Die Idee des Reduced Instruction Set Computer kam im Jahre 1980 an der University of California, Berkeley auf [4]. Man versteht darunter eine Designphilosophie für die Erstellung von Prozessorarchitekturen, bzw. deren Befehlssätzen. Das Gegenteil zum RISC ist der Complex Instruction Set Computer (CISC), welcher zu der damaligen Zeit weit verbreitet war. Dieser beruhte auf der Annahme, dass komplexe Befehle einen Geschwindigkeitsvorteil bieten, da man mit ihnen weniger Befehle benötigt um ein Programm auszudrücken. Dies hat jedoch dazu geführt, dass einzelne Instruktionen sehr viele Prozessorzyklen benötigt haben. Außerdem war es bei vielen Befehlen möglich auf den

Speicher zuzugreifen, was aber viel Zeit in Anspruch nimmt. Durch die sehr komplexen und unterschiedlichen Befehle, war es schwer diese in einer Pipeline abzuarbeiten [5]. Somit hat die wachsende Komplexität der Prozessoren letztendlich dazu geführt, dass es immer schwieriger wurde diese zu optimieren und einen Leistungsschub zu erreichen.

A. RISC-Merkmale

RISC-Architekturen sollten um einiges simpler sein. Die Idee war, dass einfache und kurze Befehle schneller abgearbeitet werden können als komplexe und lange. Die Hauptmerkmale sind dabei: (nach [4] und [5])

- Eine Load-/Store-Architektur, in der Befehle nur auf den Registern operieren. Ein Speicherzugriff ist lediglich mit den load und store Befehlen möglich, welche Daten aus dem Speicher in Register laden oder aus einem Register in den Speicher schreiben.
- Simple Instruktionen, die in einem Prozessorzyklus ausgeführt werden können.
- Eine feste Befehlslänge die das Dekodieren vereinfacht.
- Eine große Anzahl an Registern. Diese können z.B. für die Ablage von Parametern und lokalen Variablen bei einem Prozeduraufruf benutzt werden. Somit müssen keine teuren Speicheroperationen verwendet werden.

IV. Geschichte von ARM

Folgende Informationen entstammen [5]. Anfang der 1980er suchte Acorn Computers Limited - eine Computerfirma aus Cambridge, England - nach einem Nachfolgemodell für den kommerziell erfolgreichen BBC Micro. Nachdem aber kein geeigneter Prozessor gefunden werden konnte, entschied man sich eine eigene Architektur zu entwerfen. Das Problem dabei war, dass Acorn kaum Erfahrung mit Prozessordesign hatte. Es musste also eine zugleich simple, aber trotzdem leistungsstarke Architektur sein. Glücklicherweise kam etwa zur gleichen Zeit in Berkeley die Idee des RISC-Designs auf. Der erste Prozessor, der diese Philosophie umsetzte war der Berkeley RISC I. Dieser besaß gute Leistungswerte sowie eine einfache Struktur. Daher einigte man sich die gleiche Strategie zu verfolgen. 1985 begann somit die Entwicklung der Acorn RISC Machine, welche 1990 zu Advanced RISC Machine umbenannt wurde. Im gleichen Jahr wurde die Firma ARM Limited gegründet, die sich von nun an um die Entwicklung der ARM Architektur kümmern sollte.

Heutzutage vergibt ARM hauptsächlich Lizenzen an Unternehmen, die daraufhin Prozessoren basierend auf dem ARM Chip Design herstellen. Wirklich erfolgreich wurden die Prozessoren jedoch erst Mitte der 90er Jahre als die mobilen Telephone immer beliebter wurden. So antwortet John Biggs - einer der Mitbegründer von ARM Ltd. - auf die Frage, wann der Erfolg von ARM eingesetzt hat: „It really took off in the mid ’90s. That’s when the ARM7-TDMI was licensed by Texas Instruments and designed into the Nokia 6110, which was the first ARM-powered GSM phone.“ [8] Seitdem ist die ARM-Architektur vorherrschend auf dem Gebiet der Handy-Prozessoren, wird aber auch als Mikrocontroller in vielen anderen Geräten verwendet.

V. ARM Modelle

Die aktuelle Prozessor Produktreihe von ARM lässt sich in drei Kategorien einteilen [1]:

- **Cortex-A** - für Bereiche in denen hohe Leistung gefragt ist und komplexe Betriebssysteme verwendet werden.
- **Cortex-R** - verwendet in Echtzeitsystemen
- **Cortex-M** - gedacht als Mikrocontroller in eingebetteten Systemen

Für moderne Smartphones ist heutzutage hauptsächlich die Cortex-A Familie von Bedeutung. Diese teilt sich wieder in verschiedene Modelle auf, die sich insbesondere in der Leistung unterscheiden [1].

- Cortex-A5
- Cortex-A7
- Cortex-A8
- Cortex-A9
- Cortex-A15
- Cortex-A53
- Cortex-A57

Die ursprüngliche ARM Architektur hat sich mit der Zeit weiterentwickelt, sodass diese Modelle alle - bis auf A53 und A57 - mittlerweile auf der **ARMv7** Architektur basieren. Diese wird im Anschluss vorgestellt. Der Nachfolger und die momentan aktuellste Version ist die ARMv8 Architektur, welche im Ausblick vorgestellt wird. Die Cortex-A53 und -A57 Prozessoren basieren auf dieser Version, wurden aber bis heute noch nicht in einem Gerät verbaut [9].

VI. ARM Architektur

Die ARM Prozessorarchitektur ist eine 32-Bit Prozessorarchitektur. Sie wurde mit dem Gedanken entwickelt ein simples Design zu bieten, das einfache Instruktionen besitzt. Man wollte auf komplexe Befehle verzichten. Mit diesen lässt sich zwar kompakterer Code schreiben, der mit weniger Befehlen auskommt, aber die Ausführung der einzelnen Befehle dauert lang. Die meisten ARM-Befehle brauchen hingegen nur einen Prozessorzyklus zur Ausführung und sind gleich lang [5]. Daher können diese gut optimiert werden und es kann insgesamt ein Geschwindigkeitsvorteil erzielt werden.

A. Register

Die ARM-Architektur besitzt insgesamt 37 Register - alle 32 Bit groß. Der Benutzer kann auf 16 frei verwendbare Register zugreifen (R0-R15). Weiterhin ist für den Benutzer das Current Program Status Register (CPSR) sichtbar, auf das aber nicht direkt zugegriffen werden kann. Die restlichen Register sind den privilegierten Modi vorbehalten [4].

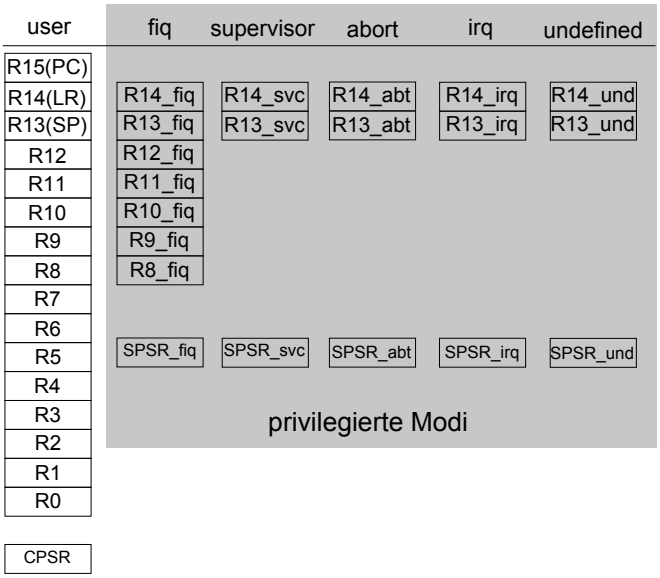


Abbildung 1. ARM Registersatz

Abb. 1 zeigt die verfügbaren Register. Auf der linken Seite stehen die Register, welche im user-Modus sichtbar sind. Dabei sind R13-R15 für bestimmte Zwecke vorgesehen. ARM empfiehlt jedoch nur diese Verwendung und theoretisch lassen sich diese Register trotzdem als Mehrzweckregister verwenden [2].

- **PC** - Program counter: In diesem Register wird die Adresse des nächsten Befehls gespeichert. Wird diese manuell verändert, so springt der Prozessor zu der neuen Adresse.
- **LR** - Link register: Hier kann bei einem Unterprogrammaufruf die Rückkehradresse gespeichert werden.
- **SP** - Stack pointer: Dieses Register wird dazu benutzt auf den Stack zu verweisen.

Abbildung 2. CPSR Register (vgl. [4] Abb. 8.3)

Das CPSR-Register (Abb. 2) dient zur Speicherung der Status- und Steuermerker. Für den Anwendungsentwickler sind dabei hauptsächlich die oberen Statusbits interessant. Diese werden bei Bedarf z.B. nach einer arithmetischen Operation oder einem Vergleichsbefehl gesetzt. Die einzelnen Bits stehen für [4]:

- **N**-Negative: Wird gesetzt wenn das Ergebnis der letzten Operation negativ war (Das oberste Bit beträgt 1).
- **Z**-Zero: Wird gesetzt wenn das Ergebnis der letzten Operation null war.
- **C**-Carry: Wird gesetzt wenn bei einer Operation Bits aus dem Register geschoben wurden oder wenn es bei einer Addition einen Übertrag gab.
- **V**-Overflow: Wird gesetzt wenn es bei einer arithmetischen Operation zu einem Überlauf kam.

Die unteren Bits des CPSR-Registers werden zur Steuerung des Prozessors benutzt. Dabei dienen die I- und F-Bits zum De-/Aktivieren der Prozessor-Unterbrechungen. Wenn das T-Bit gesetzt ist, werden die momentan ausgeführten Befehle als 16-Bit Thumb-Befehle interpretiert. Ansonsten werden die normalen ARM-Befehle angenommen [5].

Die untersten 5 mode-Bits repräsentieren den Zustand, in dem sich der Prozessor gerade befindet. Die ARM Architektur unterstützt sieben verschiedene Prozessorzustände, die im Folgenden vorgestellt werden.

B. Prozessorzustände

- **User** - Standard Modus für Anwendungsprogramme.
- **System** - Modus für Betriebssystemanwendungen.
- **Fast Interrupt (fiq)** - Für Prozessor-Interrupts, die schnell ausgeführt werden müssen.
- **Interrupt (irq)** - Für Standard-Interrupts.
- **Supervisor (svc)** - Für Software-Interrupts, die von Anwendungen angestoßen werden.
- **Abort (abt)** - Um Speicherfehler zu behandeln.
- **Undefined (und)** - Für undefinierte Befehle.

Der User-Modus ist der einzige nicht-privilegierte Modus und für normale Anwendungsprogramme vorgesehen. In ihm werden wichtige Systemresourcen vor unbefugtem Zugriff geschützt. Ein Wechsel des Zustands ist aus dem User-Modus nicht möglich [4].

Der System-Modus ist für das Betriebssystem vorgesehen. Er besitzt die gleichen Register wie der User-Modus, kann aber uneingeschränkt auf Systemresourcen zugreifen. Die restlichen Modi sind genau wie der System-Modus privilegiert und für die Unterbrechungs- und Fehlerbehandlung vorgesehen [4].

Je nach Prozessorzustand werden spezielle Register ersetzt, damit diese z.B. bei einer Unterbrechung nicht gesichert werden müssen. So werden im fiq-Modus die Register R8-R14 ersetzt. Dadurch, dass in diesem Modus nun eigene Register zur Verfügung stehen, die nicht mehr zwischengespeichert werden müssen, können Unterbrechungen von hoher Priorität schneller abgearbeitet werden. In den restlichen Modi werden lediglich R13 und R14 ersetzt [4].

Zusätzlich besitzt jeder dieser Unterbrechungs- und Fehlermodi ein Saved Program Status Register (SPSR). Dieses dient zur Speicherung des CPSR-Registers. Abb. 1 zeigt alle Register und die jeweiligen Zustände in denen sie ersetzt werden. Falls ein User-Modus-Register nicht ersetzt wird, ist es im entsprechenden Zustand auch sichtbar, sodass in jedem Zustand auf insgesamt 16 Register zugegriffen werden kann.

C. Befehlssatz

Alle Standard ARM-Befehle sind 32 Bit lang.
ARM-Prozessoren basieren, wie die meisten RISC-Prozessoren, auf einer Load/Store-Architektur. Das bedeutet, dass auf Daten aus dem Hauptspeicher nur über entsprechende Lade(load)- und Speicher(store)-Befehle zugegriffen werden kann. Sämtliche Befehle, die Daten verändern oder analysieren können somit nur mit direkten Werten oder Registern arbeiten. Es ist nicht möglich, wie z.B. bei der Intel-Architektur Speicheroperanden anzugeben [5].

Dabei lassen sich bei vielen Befehlen bis zu drei Register angeben. So können das Zielregister und beide Operandenregister unabhängig voneinander bestimmt werden.

Weiterhin ist es möglich jeden Befehl konditioniert ausführen zu lassen. Das heißt, es ist möglich eine Bedingung mit anzugeben, welche vor dem Befehl überprüft wird. Falls diese gegeben ist, wird der Befehl ausgeführt, ansonsten wird zum nächsten Befehl übergegangen. Dadurch lassen sich zusätzliche Compare- und Branch-Befehle einsparen, die z.B. bei der Intel-Architektur zur Auswertung Boolscher-Ausdrücke nötig wären [5].

Viele Befehle, wie z.B. `mov` unterstützen ein Verschieben oder Rotieren des zweiten Operandenregisters, bevor der Befehl ausgeführt wird. Daher werden keine gesonderten Befehle zum *shiften* benötigt.

Eine vollständige Übersicht über die wichtigsten Befehle findet sich auf der Instruction Set Reference Card von ARM [3]. Allgemein lässt sich der Befehlssatz in drei Kategorien unterteilen [5]:

- Datenverarbeitungsbefehle: Befehle, die Daten aus Registern verändern oder auswerten. Z.b. Arithmetische und logische Funktionen, sowie die `mov`-Befehle.
- Datentransferbefehle: Befehle, die Daten aus dem Speicher in Register laden oder umgekehrt.
- Kontrollflussbefehle: Befehle, die zur Steuerung und Verzweigung des Programms dienen.

Im Folgenden sollen auf die Datenverarbeitungsbefehle näher eingegangen und ein paar Befehle vorgestellt werden.

D. Datenverarbeitungsbefehle

Alle Datenverarbeitungsbefehle operieren ausschließlich auf Registern. Als arithmetische Funktionen stehen `add`, `sub`, und `mul` zur Verfügung, zum Addieren, Subtrahieren und Multiplizieren. Es gibt **keinen** direkten Befehl zum Dividieren. Die Syntax des `add`-Befehls ist folgende [4]:

```
add{cond}{s}        Rd, Rn, operand2
```

Dabei gibt `Rd` das Zielregister der Operation und `Rn` das erste Operandenregister an. `operand2` kann entweder ein Register oder ein direkter Wert sein. Wird ein Register angegeben, so kann dieses entweder über einen direkten Wert oder über ein weiteres Register verschoben, bzw. rotiert werden (s. Tab. *Flexible Operand 2* in [3]).

Da jeder Befehl konditioniert ausgeführt werden kann, ist es möglich an das Mnemonic ein Bedingungskürzel (`cond`) anzuhängen (s. Tab. *Condition Field* in [3]).

Zusätzlich lässt sich mit einem `s` am Ende des Befehls ausdrücken, dass die Statusmerker gesetzt werden sollen. Wird dieses weggelassen, so wird der Wert in den Merkern nicht überschrieben, wodurch sich Ergebnisse einer Operation über mehrere Befehle hinweg speichern lassen.

Folgendes Beispiel zeigt eine bedingte Multiplikation mit 3 mittels `add`:

```
addeqs        R0, R1, R1, lsl #1
```

Durch diesen Befehl wird zunächst überprüft ob das Zero-Flag gesetzt ist (`eq`). Falls dies zutrifft, wird der Wert des Registers R1 um 1 nach links verschoben (`R1, lsl #1`), mit sich selber addiert und dann in R0 gespeichert. Zusätzlich werden die Statusmerker gesetzt.

Die gleiche Syntax (bis auf den Befehl selber) besitzen die logischen Funktionen, wie z.B. `and` oder `orr`. Diese arbeiten bitweise und speichern das Ergebnis der Operation im Zielregister ab. Zusätzlich gibt es die `cmp`- und `tst`-Befehle, welche für logische Vergleiche von Registern vorgesehen sind. Im Vergleich zu `and` werden hier jedoch nur die Statusmerker implizit gesetzt und das Ergebnis verworfen, weshalb kein Zielregister angegeben wird [3].

Zum Verschieben von Werten zwischen Registern gibt es den `mov`-Befehl, bzw. `mvn` zum Verschieben des negierten Wertes. Diese Befehle kommen mit einem Operand aus, welcher wie `operand2` frei gewählt werden kann.

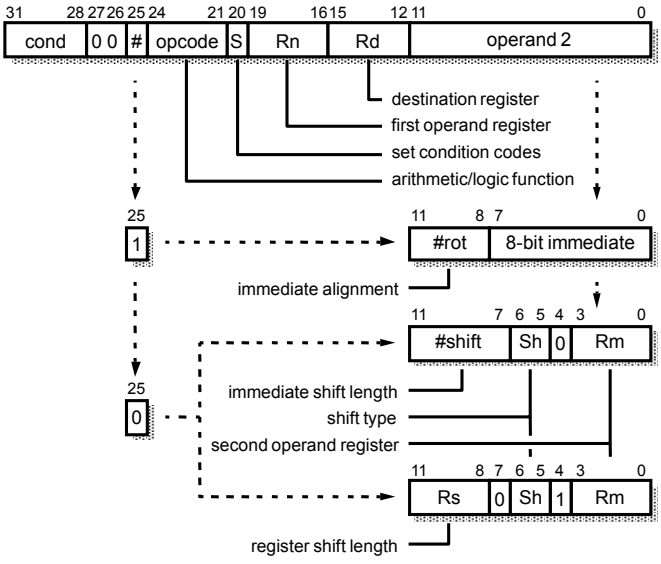


Abbildung 3. Kodierung der Datenverarbeitungsbefehle (siehe [5] Abb. 5.6)

Abb. 3 zeigt die Befehlskodierung, die für alle Datenverarbeitungsbefehle verwendet wird, außer der Multiplikation. Die obersten 4 Bits repräsentieren die Bedingung, mit welcher der Befehl ausgeführt wird. Die darauffolgenden 2 Bits sind in dieser Kodierung auf 0 gesetzt. Das 25. Bit signalisiert, ob als zweiter Operand ein direkter Wert (1) oder ein Register genommen wird (0). Die opcode-Bits geben die jeweilige Funktion an, die auf die Operanden angewandt werden soll. Das S-Bit bestimmt, ob die Statusmerker gesetzt werden. Darauf folgen das Zielregister und das erste Operandenregister. Die letzten 12 Bits bestimmen sich je nach Art des zweiten Operanden.

Wird ein direkter Wert als zweiter Operand verwendet, so wird dieser aus einem direkten 8-Bit-Wert und einem 4-Bit-Rotationswert gebildet. An den Rotationswert wird eine 0 von unten angehängt, damit er 5 Bit lang ist. Dadurch entstehen nur gerade Rotationswerte. Durch Rotation des 8-Bit-Wertes wird nun der ursprünglich angegebene Wert gebildet. Mit diesem Verfahren können nicht alle Zahlen von 0-4096 abgedeckt werden, wie es mit einem direkten 12-Bit Wert möglich wäre. Jedoch deckt es einen großen Bereich ab, z.B. alle 2er Potenzen, die sich mit 32-Bit darstellen lassen.

Die zweite Möglichkeit ist ein Register Rm als zweiter Operand. Hierbei ist es entweder möglich, direkt einen 5-Bit-Wert anzugeben (#shift) oder es wird ein Register Rs angegeben, um dessen Wert das Register verschoben wird. Mit den Sh-Bits wird letztendlich die Art der Verschiebung, bzw. Rotation kodiert.

E. Thumb-Befehlssatz

Die ARM-Architektur unterstützt zusätzlich zu den normalen 32-Bit ARM-Befehlen den Thumb-Befehlssatz. Dieser wurde erstellt um die Code-Dichte zu erhöhen, da alle Thumb-Instruktionen nur 16-Bit groß sind. Der Prozessor wandelt diese bei der Ausführung in äquivalente 32-Bit-Befehle um. Es ist ebenfalls möglich beide Befehlssätze zu kombinieren, da durch das T-Bit im CPSR-Register angegeben ist, wie der Prozessor die eingelesenen Befehle interpretieren soll. Ist das Bit gesetzt, wird der Thumb-Befehlssatz angenommen, sonst der normale ARM-Befehlssatz. Die wichtigsten Unterschiede bei den Thumb-Befehlen sind [5]:

- Es ist nicht möglich die Befehle - bis auf den *branch*-Befehl - konditioniert ausführen zu lassen.
- Die meisten Befehle besitzen ein 2-Adress-Format. D.h. das Zielregister ist gleichzeitig Operandenregister.
- Es hängt von den Befehlen ab, ob die Statusmerker gesetzt werden, oder nicht.
- Ein unregelmäßigeres Befehlsformat, aufgrund der geringen Befehlslänge.
- Die meisten Befehle benutzen nur die Register R0-R7 und R13-R15

F. big.LITTLE

Durch das big.LITTLE-System strebt ARM eine hohe Leistung bei gleichzeitig geringem Energieverbrauch ihrer Prozessoren an [6]. Hierbei werden zwei Prozessorkerne derselben Architektur, aber mit unterschiedlichen Leistungswerten, in einem System vereint. Momentan wird diese Technik bei den Cortex-A15 und Cortex-A7 Prozessoren angewandt. Der A15 besitzt eine hohe Leistung, aber auch einen hohen Energiebedarf. Der A7 hingegen arbeitet sparsamer, aber auch langsamer. Beide Kerne unterstützen die ARMv7-Architektur und können somit dieselben Befehle ausführen. Diese Tatsache wird ausgenutzt, indem die Programme unterschiedlich auf die Prozessorkerne verteilt werden, je nachdem ob die Bearbeitung hohe oder niedrige Priorität hat. Somit steht dem System nur soviel Leistung zur Verfügung wie es gerade benötigt, was vor allem bei mobilen Geräten einen großen Vorteil bietet, aufgrund der gesteigerten Energieeffizienz [6].

VII. Ausblick

A. ARMv8

Am 31.10.2012 wurde von ARM Ltd. der offizielle Nachfolger der ARMv7-Architektur - die ARMv8 - angekündigt [9]. Die größte Neuerung ist, dass erstmals auf eine 64-Bit-Architektur umgestiegen werden soll. Abwärtskompatibilität zu den 32-Bit-Architekturen soll aber trotzdem beibehalten werden. Die ersten Prozessoren, die diese Architektur unterstützen, sind der Cortex-A53 und Cortex-A57, welche auch zusammen durch das big.LITTLE-Verfahren betrieben werden können. ARM verspricht sich mit den neuen Prozessoren, welche in 20-nm-Versionen gefertigt werden sollen eine Leistungssteigerung von bis zu 50 % im Vergleich zu den Cortex-A15 Prozessoren [9]. Zunächst sollen diese vor allem im Server-Bereich verwendet werden. Es ist aber durchaus denkbar, dass sie auch in modernen High-End-Smartphones zum Einsatz kommen [7].

Literatur

[1] ARM Ltd. (2013): ARM Processors. http://www.arm.com/products/processors/index.php (03.02.2013)

[2] ARM Ltd. (2012): ARM Architecture Reference Manual ARMv7-A and ARMv7-R edition. Cambridge, England

[3] ARM Ltd. (2008): ARM and Thumb-2 Instruction Set Quick Reference Card. http://infocenter.arm.com/help/topic/com.arm.doc.qrc0001m/QRC0001_UAL.pdf (03.02.2013)

[4] Dandamudi, Sivarama P. (2005): Guide to RISC Processors. New York, USA: Springer

[5] Furber, Steve (2000): Arm System-On-Chip Architecture 2nd ed. Harlow, England: Addison-Wesley

[6] Greenhalgh, Peter (2011): Big.LITTLE Processing with ARM Cortex-A15 & Cortex-A7. http://www.arm.com/files/downloads/big_LITTLE_Final_Final.pdf (03.02.2013)

[7] Nichols, Scott (2012): ARM Cortex-A50 chips to power future 64-bit smartphones http://www.techradar.com/news/computing-components/processors/arm-cortex-a50-chips-to-power-future-64-bit-smartphones-1108796 (03.02.2013)

[8] Sakr, Sharif (2011): The Engadget Interview: ARM co-founder John Biggs. http://www.engadget.com/2011/12/20/the-engadget-interview-arm-co-founder-john-biggs/ (17.01.2013)

[9] Windeck, Christof (2012): ARM stellt 64-Bit-Prozessorkerne Cortex-A53 und -A57 vor http://heise.de/-1740587 (03.02.2013)

HW-Sensoren und -aktoren moderner Handies (d.h., insb. E/A-Geräte)

Tim Auf dem Berge

***Abstract*—Diese Ausarbeitung beschäftigt sich mit den am häufigsten in modernen Handys verbaute Sensoren und Aktoren. Dabei wird auf die unterschiedlichen Displaytechnologien wie LCD und OLED, sowie die zwei verbreitesten Touchtechnologien Kapazitiv und Resistiv eingegangen. Weiterhin beschäftige ich mich mit den wichtigsten Sensoren, wie denen zur Erfassung von Bewegungseingaben. Dazu gehören zum Beispiel der Beschleunigungssensor und das Gyroskop. Darüber hinaus geh ich auf einige in modernen Handys verwendete Aktoren ein, zum Beispiel einen linearen Ultraschall Piezomotor, welcher in dem Autofokus Modul von Smartphone Kameras verwendet wird.**

***Keywords*-Handys; Smartphones; Sensoren; Aktoren; Touchscreen;**

I. Sensoren

Sensoren sind Messinstrumente welche thermische, mechanische, biochemische, magnetische oder strahlungsbedingte Größen messen, in einen elektrischen Strom umwandeln und damit als Schnittstelle zur Umgebung dienen. Diese Messgrößen werden weiter an auswertende Module übermittelt und von Diesen verarbeitet. Für elektrische Systeme sind Sensoren die ”Sinnesorgane künstlicher Systeme“. [Fat04]
Im Folgenden werde ich auf einige der wichtigen Sensoren die in Smartphones verbaut werden eingehen und einen Überblick über weniger verbreitet Sensoren geben.

A. Touchscreen

Der wohl wichtigste Sensor in modernen Handys ist der Touchscreen der zur Ein- und Ausgabe verwendet wird. Aktuell sind zwei verschiedene Arten von Displaytechnologien verbreitet. Zum einen ist das die Liquid Crystal Displays (LCD) Technologie und zum anderen die Organic Light Emitting Diode (OLED) Technologie.
Bei der LCD Technologie bestehen die Pixel aus Flüssigkristallen in den Farben Blau, Grün und Rot. Die Flüssigkristalle sind verdreht und können sich abhängig von einem Stromfluss zu einem bestimmen Grad entdrehen und dadurch durchfallendes Licht beeinflussen. Dieses Licht kommt von einer Hintergrundbeleuchtung die durch einen Polarisationsfilter gesteuert wird.
Die einzelnen Pixel werden über einen Dünnschichttransistor (TFT) angesteuert, welcher in einer Matrix über den Pixeln angeordnet ist. Über die Regulierung des Stroms, kann die Intensität der blauen, grünen und roten Subpixel gesteuert werden um den Pixel in einer bestimmten Farbe erscheinen zu lassen. [HSW01]
Bei der OLED Technologie werden organische Polymere oder kleine organische Moleküle eingesetzt, weshalb sie als Organic LED bezeichnet werden. OLEDs benötigen keine Hintergrundbeleuchtung oder Filter und können deshalb wesentlich flacher ausfallen als LCDs. OLEDs bestehen aus verschiedenen Schichten. Es gibt eine Schicht die als Anode fungiert und eine die als Kathode fungiert. Zwischen diesen Schichten gibt es eine Übertragerschicht und eine Emmisionsschicht. Wenn Strom durch die Schichten fließt, phosphoresziert die Emmisionsschicht und leuchtet. Die Farbe der Schicht ist abhängig von den verwendeten organischen Materialien. Deshalb werden bei Farbdisplays mehrere dieser Schichten verwendet. Die Intensität des Lichts kann durch den Stromfluss gesteuert werden. Um die Pixel anzusteuern wird wieder eine Matrix mit TFT verwendet, weshalb auch der Begriff Active Matrix OLED (AMOLED) für diesen Typ verwendet wird. [HSW02]
Unabhängig von der verwendeten Displayvariante besitzen die meisten Modernen Handys einen kapazitiven Touchscreen. Hierbei wird oberhalb des Displays eine Schicht angebracht, welche elektrische Ladung speichert. Oberhalb davon befinden sich kratzfeste Schichten die das Display schützen sollen. Wenn nun ein Finger das Display berührt wird ein Teil der elektrischen Ladung in den Finger übertragen, sodass sich die Ladung in der Kapazitiven Ladungsschicht reduziert. Diese Reduzierung der Ladung wird im Stromkreis festgestellt welcher in den Ecken des Displays gemessen wird. Durch die unterschiedlichen Ladungen in den Ecken kann bestimmt werden an welcher Stelle der Finger das Display berührt hat. [HSW03]
Eine andere Methode Touch-Eingaben des Benutzers zu erkennen, ist die Verwendung eines resistiven Touchscreens. Bei dieser Technik befinden sich oberhalb einer Glasscheibe zwei metallische Schichten. Bei den beiden Schichten handelt es sich um zwei leitfähige transparente Folien. Darüber befindet sich zum Schutz noch eine kratzfeste Schicht. Die beiden metallischen Schichten sind durch nicht leitende Abstandshalter voneinander getrennt. Entsteht jetzt ein Druck auf das Display, berühren sich die beiden Schichten und leiten Strom. [HSW03] Da dort eine Gleichspannung angelegt ist, welche an den Rändern gemessen wird, kann durch die unterschiedlichen Spannungen bestimmt werden wo sich die beiden Schichten berühren. Dazu wird erst der Strom auf der X-Achse ausgewertet und einiger Millisekunden später wird eine Spannung auf der Y-Achse angelegt und dort die Position bestimmt. Je höher die Spannung

am Messpunkt ist, desto weiter ist der Druckpunkt vom Messpunkt entfernt.
Der Vorteil von resistiven Touchscreens gegenüber kapazitiven Touchscreens ist, dass zur Eingabe auch ein Eingabestift verwendet kann oder der Benutzer Handschuhe tragen kann, da das Objekt das den Druck erzeugt nicht leitend sein muss. Kapazitive Touchscreens funktionieren nur mit Eingaben mit dem nackten Finger, was besonders im kalten Winter zum Nachteil wird. Ein weiterter Unterschied der beiden Techniken ist, dass resistive Touchscreens bei der Eingabe mit dem Finger nicht so genau sind wie kapazitive Touchscreens. [CO01] Kapazitive Touchscreens sind häufiger verbreitet, da sie Multitouch unterstüzen, also die gleichzeitige Eingabe mit mehreren Fingern und weil sie Lichtdurchlässiger sind als resistive Touchscreens.

B. Beschleunigungssensor

Der Beschleunigungssensor oder Accelerometer ist ein Sensor der die Beschleunigung des Smartphones misst. Hierbei wird die Beschleunigungsmessung auf drei Achsen vorgenommen (X-, Y- und Z-Achse).
Häufig verwendet werden piezoelektronische Beschleunigungssensoren. Dabei werden Piezoelemente, welche eine Widerstandsänderung hervorrufen, wenn mechanischer Druck auf sie einwirkt, verwendet. Diese Widerstände werden an den Stellen eines Biegebalkens angebracht an denen die Verformungskräfte wirken. Der Biegebalken wird mit einer seismischen Masse beschwert und wenn nun durch Beschleunigung sich der Balken biegt führt das zu einer messbaren Spannungsänderung durch den Piezowiderstand. [Fat04]
Andere Methoden dieses Sensor zu realisieren ist durch ein Piezoelement welches sich unter einem durch eine Feder gehaltenen seismischen Masse befindet. Durch Beschleunigungseinwirkung wird Druck auf das Piezoelement eingewirkt und dass führt dann zu einer messbaren Änderung des Widerstandes. [SHW03]
Der Beschleunigungsbereich des Sensors der z.B. Im Apple Iphone 4 verwendet wird beträgt +/- 2g mit einer Sensitivität von 0,018g.
Als Referenzwert beträgt die normale g Einwirkung auf Meereshöhe 1g und zum Beispiel auf ein von 0 auf 100 km/h (2,4s) beschleunigender Bugatti wirken 1,55g. [SHW03]
Die Abmessung des im Iphone 4 verwendeten LIS331DLH Accelerometers beträgt 3mm x 3mm x 1mm. [MEMS1]

In Figur 1 ist das LIS331DLH Accelerometer, welches im Iphone 4 verwendet wird abgebildet. Die mit der Nummer 1 gekennzeichneten Balken sind die Federn für die entsprechend gekennzeichneten Achsen. In den eingerahmten Bereichen befinden sich die für den Piezoelektrischen Effekt benötigten seismischen Massen. Die X-, Y- und Z-Achse sind durch die jeweiligen Buchstaben

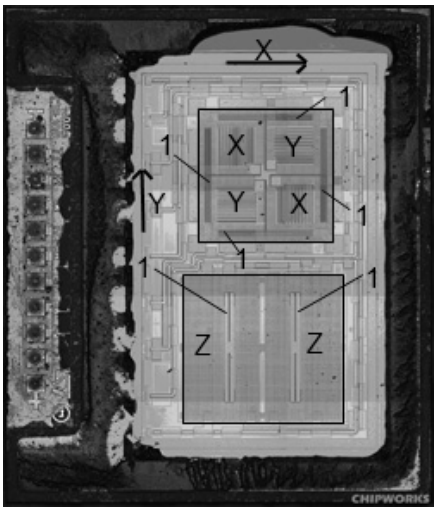


Figure 1. Vergrößerung des Iphone 4 Beschleunigungssensors [CW01]

gekennzeichnet.
Der wohl offensichtlichste Nutzen eines Beschleunigungssensors in Smartphones ist der Wechsel zwischen Hochformat und Querformat des Bildschirms. Weiterhin verwenden aber auch viele Spiele-Apps den Sensor um über Neigung des Smartphones das Spiel zu Steuern.

C. Gyroskop

Ein Gyroskop misst die Ausrichtung des Smartphones im Raum und erkennt Änderung dieser. Ergänzend zum Beschleunigungssensor erkennt das 3-Achsen-Gyroskop Rotationen um die X-, Y- und Z-Achsen. Damit sind viele Geräte mit einer sechs Achsen Messung ausgestattet.
Die in Smartphones verwendeten Gyroskope sind nicht die wie im Flugzeug verbauten Kreiselinstrumente mit einem drehen Kreisel, der seine Ausrichtung zur Erdoberfläche beibehält, sondern funktionieren wie Vibrationskreisel. Auf dem Chip befinden sich vier Prüfmassen mit Feder, an welchen über Kondensatoren Verschiebungen gemessen werden können. Diese Prüfmassen schwingen und werden über ein ”drive“ Signal gesteuert. Wenn sich nun durch Drehung eine Corioliskraft auf die Prüfmassen auswirkt, werden diese abhängig von der Drehrichtung nach Außen gedrückt. Diese Verschiebung wird dann über Spannungsänderung von einer integrierten Schaltung ausgewertet und dann an den Prozessor weitergeleitet. [ifix1]
Die Abmessung des Iphone 4 Gyroskops beträgt 4mm x 4mm x 1mm. [MEMS2] Das Gyroskop findet Verwendung in Spiele-Apps und unterstützend für GPS wenn zum Beispiel ein Tunnel durchfahren wird und kein Satelliten Signal zu Verfügung steht. Dann kann über das Zusammenspiel von Beschleunigungssensor und Gyroskop weithin die Position des Smartphones bestimmt werden. Der elektronische Kompass kann diese Positionierung noch verbessern.

D. Elektronischer Kompass

Der elektronische Kompass oder Magnetometer ist ein Gerät das die Stärke von Magnetfeldern ermitteln kann. Elektronische Kompasse werden mit Hilfe von Hall-Sensoren realisiert. Die Magnetometer messen die Felder auf drei Achsen. Durch Zusammenspiel vom elektronischen Kompass, Gyroskop und Beschleunigungssensor kann auch innerhalb von Gebäuden oder Tunnel navigiert werde, wenn kein GPS Signal vorhanden ist. [MEMS3]

E. GPS

Die meisten Smartphones sind heutzutage mit einem Global Positioning System Empfänger (GPS-Empfänger) ausgestattet. Beim GPS wird anhand von Zeitmessung von Signalen die Distanz zu einem Satelliten gemessen. Durch diesen Wert lässt sich aber noch nicht der genaue Standort des Empfängers ermitteln. Erst durch Entfernungsmessung zu mindestens drei Satelliten lässt sich ein die Position des Empfängers auf der Erde erfassen. Hierzu wird anhand der Entfernungsmessung der Satelliten von jedem ein Radius gezogen in welchem sich der Empfänger befindet. An der Schnittstelle der Radien kann nun ein Bereich bestimmt werden in dem Sich der Empfänger befindet. [GPS01]
Da dieses Berechnung aber in guten Fällen mindestens 40 Sekunden benötigt, wird zur Unterstützung das A-GPS Assisted Global Positioning System genutzt um diese Zeit zu verkürzen. Um verfügbare GPS-Satelliten zu finden werden Informationen über diese per Handynetz übertragen oder der vom Handy verwendete Sendemast übermittelt seinen Standort. [CB01] Übliche Genauigkeitswerte von GPS zur Bestimmung der Position sind 20m bis 50m. [SHW03] GPS in Smartphones wird überwiegend zum Navigieren mit Hilfe von Navigations-Apps verwendet. Außerdem werden anhand des Standortes nützliche Information zur Umgebung aus dem Internet gesammelt und dem Benutzer zur Verfügung gestellt. Beispiele hierfür sind Restaurant Empfehlungen oder Hintergrundwissen über Sehenswürdigkeiten.

F. Näherungssensor

Näherungssensor werden in Smartphones verwendet um zu erkennen, ob sich Objekte nah vor dem Sensor befinden, wie zum Beispiel ein Ohr während des Telefonierens. Der Näherungssensor ist ein kleiner Sensor der auf der Vorderseite des Smartphones oberhalb des Touchscreens sitzt. Wenn er aktiviert ist, schickt eine kleine Diode Infarotsignale los und misst anhand des Zeitunterschieds zwischen versenden des Signals und wieder Eintreffen des an einem Objekt zurückgeprallten Signals, den Abstand zum jeweiligen Objekt. Diese Funktion wird benutzt um beim Telefonieren den Touchscreen auszuschalten, während sich das Smartphone am Ohr des Benutzers befindet, um ungewünschte Eingaben zu verhindern. [YT01]

G. Umgebungslichtsensor

Der Umgebungslichsensor oder Helligkeitssensor misst die Helligkeit in des Lichts das auf den Sensor trifft. Genau wie der Näherungssensor ist der Helligkeitssensor auf der Vorderseite des Smartphones oberhalb des Touchscreens angebracht. Der Sensor misst über ein Photometer die für das menschliche Auge erkennbare Helligkeit und sendet diesen Wert an den Prozessor. [AL01] Häufiger Nutzen des Umgebungslichtsensors ist die Beleuchtung der Funktionstasten des Smartphones an- bzw. auszuschalten oder die Displayhelligkeit zu regulieren.

H. Fallsensor

Der Fallsensor ist ein wenig verbreiteter Sender, welcher nur in Handys verbaut ist die für Senioren gedacht sind. Fallsensoren sind ebenfalls Bewegungsensoren, jedoch besitzen sie nicht den Umfang von den 3-Achsen Bewegungssensoren welche in Smartphones verwendet werden. Diese Sensoren erkennen einen Sturz des Benutzers und können diesen von einem fallen lassen des Handys unterscheiden. Im Falle eines Sturzes des Benutzers wird eine Notruf SMS oder ein Notruf-Anruf verschickt. Diese Funktion wird häufig in Kombination mit Standortinfomation über den Benutzer versandt, welche über GPS ermittelt wurden.[IH01]

I. Mirkofon

Die in den Handys verwendeten Mikrofone fallen in den Bereich der Mikrosensoren. Sie basieren auf dem gleichen Prinzip wie andere Mikrofone, indem sie Schalldruck in elektrisch Signale umwandeln, welche dann als Geräusche beziehungsweise Sprache ausgewertet werden können. Diese Mikrofone besitzen eine extrem dünne Membran, welche die Schallsignal aufnimmt und piezoelektrisch oder kapazitiv auswertet.
Der Stromverbrauch dieser Mikrofone liegt bei unter 1mW und sie filtern Störgeräusch überwiegend raus. [IT01]

In neuen Smartphones wie dem Iphone 5 sind drei Mikrofone verbaut um die Spracheingaben des Benutzers aus unterschiedlichen Winkeln aufzunehmen und die Eingabe qualitativer auszuwerten. [IN01]

II. Aktoren

Aktoren sind das Gegenstück zu Sensoren, denn Sensoren wandeln mechanische Energie in Strom um und Aktoren wandeln elektrischen Strom in mechanische Energie um. Für die Verwendung in modernen Handys kommen größtenteils nur Mikroaktoren in Frage, da die neuen Handys kontinuierlich mit weiteren Funktionen erweitert werden, dabei der Platz für Module auf der Leiterplatte nicht zunimmt. Deshalb müssen die verbauten Mikrochips immer kleiner und leistungsfähiger werden. Aktuelle Mikrochips bewegen sich im Größenbereich von einigen Millimetern. Im Folgenden werde ich mich mit den in Handys verwendeten Aktoren auseinander setzen.

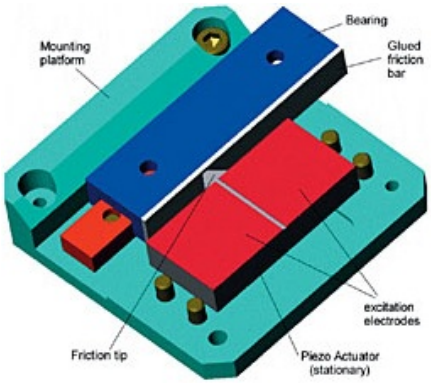


Figure 2. Schema eines Linearen Ultraschall Piezomotors [PI01]

A. Auto-Fokus Aktor

Der Auto-Fokus Aktor einer Kamera für Smartphones ist meistens im Kameramodul integriert. Zusätzlich zu der Kamera und dem Auto-fokus Aktor gehört häufig auch noch eine LED Lampe dazu, welche Aufnahmen in dunklerer Umgebung ermöglichen. Dieses Modul hat circa eine Abmessung von 10mm x 10mm x 8 mm. [STM01] Bei dem Auto-Fokus Aktor handelt es sich um einen Piezoelektronischen Mikroaktoren der in Stapelbauweise zusammengesetzt wurde. Dazu werden dünne Schichten von Piezoelementen und Metallelektroden übereinander gesetzt. [Fat04] Piezoelemente haben die Eigenschaft mechanische Energie in elektrische Energie umzuwandeln, den direkten piezoelektronischen Effekt. Wird zum Beispiel ein Druck auf das Piezoelement ausgeübt entsteht eine elektrische Spannung. Umgekehrt gibt es auch den inversen piezoelektronischen Effekt dabei führt das Anlegen einer Spannung dazu, dass das Piezoelement seine Länge und Dicke ändert.
Die Stapelbauweise erhöht die Längen und Dickenänderungen der Struktur im Vergleich zu einem großen Piezoelement. Durch die piezoelektronischen Effekte dient der Piezoaktor gleichzeitig auch als Sensor und macht einen solchen überflüssig. [Fat04]
Für das Auto-Fokus Modul wir ein Piezomotor genutzt, welcher senkrecht zum Objektiv einen vibrierenden Ultraschall Strahl produziert. Diese ist an zwei Stellen mit der Linse verbunden um die Stabilität und Antriebskraft zu maximieren. Die für den Piezomotor benötigte Spannung wird direkt aus dem Smartphone-Akku genommen wird. Hierfür reichen bereits weniger als 2.8V aus. [DI01]

In Figur 2 ist ein Schema eines linearen Ultraschall Piezomotors abgebildet, welcher mit Hilfe einer Keramik Spitze funktioniert. Diese Spitze wird durch die hochfrequente Eigenschwingungen von Zehn bis Hunderten von Kilohertz bewegt, indem sich die rot abgebildete Masse verformt, in welcher sich die Spitze in der Mitte befindet. Durch diese Bewegungen wird der blau dargestellte bewegliche Teil verschoben. Durch zwei Elektroden, welche links und rechts der Masse angebracht sind kann die Richtung der Schwingungen beeinflusst werden. Dieses bestimmt die Richtung in welche der bewegliche Teil verschoben wird. Mit jedem Mikroimpuls wird der bewegliche Teil einige Nanometer bewegt. Diese Mikrobewegungen bilden einen flüssigen Bewegungsablauf. [PI01]
Die Vorteile eines Piezoelektronischen Mikroaktors sind die präzise Bewegungen im Nanometerbereich und die schnellen Ansprechzeiten im Mikrosekundenbereich. [Fat04] Diese präzise Bewegung ist sehr wichtig für die exakte Fokussierung der Kamera auf ein Objekt. Ein Unterschied von diesen linearen Ultraschall Piezoaktoren gegenüber herkömmlichen Linearmotoren ist, dass kein Stromverbrauch entsteht um die Linse in ihrer Position festzustellen, aber sie auch nicht die unbegrenzte Bewegungsreichweite von linearen Piezomtoren haben. Lineare Ultraschall Piezomotoren sind auf eine Strecke von ca. 50nm begrenzt, beschleunigen dafür aber deutlich schneller als normal linear Piezomotoren. [pw08]

B. Vibrationsmotor

Der Vibrationsmotor oder auch Vibramotor genannt ist ein Motor, welcher durch Schwingungen das Handy zum Vibrieren bringt. Dieses wird häufig als Benachrichtigung oder Alarmmeldung verwendet. Es gibt Vibrationsmotoren die als Rotationsmaschinen mit Unwucht Gewichten realisiert sind. Hierbei ist ein Gewicht mit einer starken Unwucht an einer Seite um eine Achse positioniert. Wenn nun der Motor das Gewicht um die Achse dreht, gerät der Motor in Schwingungen, was sich auf das Gehäuse auswirkt. Die Vibrationsmotoren werden häufig in der Nähe des Rahmens positioniert.
Eine andere Methode Vibartionsmotoren zu realisieren ist mit linear schwingenden Vibrationsmotoren. Hierbei wird mit Hilfe einer Feder und eines Magneten eine Masse in Bewegung gebracht um dadurch eine Schwingung zu erzeugen. Der Vorteil von linear schwingenden Vibrationsmotoren gegenüber Rotationsmotoren mit Unwucht Gewichten ist, dass sie im Vergleich leiser sind. Jedoch können die linear schwingenden Vibrationsmotoren größer als die Rotaionsmotoren ausfallen. [ifix2]
Der Vibrationsalarm am Handy kann anstatt des klassischen Klingeltons verwendet werden, um den Benutzer über eingehende Anrufe oder Mitteilungen zu informieren. Bei vielen Smartphones wird auch eine kleine Vibration als Bestätigung für Eingaben auf den Touchscreen verwendet.

C. Lautsprecher

In modernen Handys sind mehrere Lautsprecher verbaut. Zum einen ein Lautsprecher oberhalb des Displays, welcher zum Telefonieren verwendet wird und zum anderen ein Lautprecher auf der Rückseite welcher für Soundausgaben bei

Musik und Videowiedergabe sorgt. Bei dem Lautsprecher zum Telefonieren handelt es sich um ein ganzes Lautsprechermodul, dass mit einer Antenne und einem Mikrofon gekoppelt ist.

III. MEMS

MEMS steht für Micro-Electro-Mechanical Systems und sind Chips die Logikelemente und mikromechanische Strukturen verbinden. Im Deutschen werden MEMS auch als Mikrosystemtechnologie (MST) bezeichnet. Diese Chips finden oft Verwendung als Sensoren und Aktoren und werden meist aus Silizium hergestellt. Ein MEMS kann grob unterteilt werden in einen Aktorteil, einen Sensorteil und einen Signalverarbeitungsteil. Dank der heutigen Technik lassen sich diese Millimeter großen Chips mit den Mikrometer großen Strukturen in Massen herstellen. Die Struktur von MEMS kann stark variieren, von simplen Anordnungen ohne bewegliche Teile, bis hin zu sehr komplexen Strukturen mit mehreren beweglichen Teilen. Sensoren und Aktoren die als MEMS realisiert sind übertreffen ihre Gegenstücke in größeren Ausführungen häufig durch erhöhte Genauigkeit und Leistung. [MN01] MEMS lassen sich ebenfalls mit den üblichen CMOS (Complementary Metal Oxide Semiconductor) Herstellungsverfahren produzieren. Der Unterschied zu CMOS ist, dass viele MEMS mehrere übereinander liegende Chips besitzen können und das aufwendigere dreidimensionale Strukturen, eine Fertigung durch zum Beispiel Ätzverfahren benötigen. [EK01]

IV. Zusammenfassung

Die Sensoren und Aktoren sind die Sinnesorgane und Gliedmaßen der modernen Handys. Durch ihre umfangreichen Sensoren und hochauflösenden Displays sind moderne Touchscreenhandys, welche über ausreichend leistungsfähige Hardware besitzen sogar Konkurrenten zu tragbaren Spielekonsolen geworden. Durch Verwendung der vielseitigen und präzisen Bewegungsensoren werden auch immer mehr Spiele-Apps entwickelt die eine Bewegungssteuerung nutzen.
GPS und die unterstützenden Bewegungssensoren machen Smartphones zu einer ebenbürtigen Alternative zu Navigationsgeräten. Darüber hinaus werden auch die schon länger verwendeten Funktionen wie Kamera und Telefonieren durch immer bessere Mikrofone und Kameras verbessert. Der Trend von Smartphones geht dazu immer mehr und leistungsfähigere Sensoren zu verwenden. Dabei werden die Geräte aber nicht größer und deshalb müssen auch die Mikroaktoren leistungsfähiger und umfangreicher werden, ohne das die Mikrochips größer werden.
Neue Sensoren werden überwiegend die Bewegungssensoren haben ihren Ursprung in der Luft und Raumfahrt. Wurde größere mechanische Gyroskope früher in Flugzeugen verwendet um die Neigung des Flugzeugs zum Horizont zu bestimmen, so sind passen diese Sensoren heutzutage als Mikrochips in Smartphones.
Mit der heutigen Technologie zur Herstellung von Mikrochips insbesondere MEMS sind nahezu alle möglichen Sensoren und Aktoren auch als Mikrosensoren und Mikroaktoren realisierbar. Eventuell werden zukünftige Handys sogar in der Lage sein Temperaturen zu messen oder die Luft zu analysieren. In Naher Zukunft sind diese Chips vielleicht sogar als Nanosensoren und Nanoaktoren denkbar.

References

[Fat04] Prof. Dr.-Ing. S. Fatikow: *Skript zur Vorlesung Mikrorobotik und Mikrosystemtechnik*. Universität Oldenburg, 5. Auflage, 2004.

[HSW01] How Stuff Works *How LCDs Work*. http://electronics.howstuffworks.com/lcd2.htm, Letzter Zugriff: 2013-02-02.

[HSW02] How Stuff Works *How OLEDs Work*. http://electronics.howstuffworks.com/oled1.htm, Letzter Zugriff: 2013-02-02.

[HSW03] How Stuff Works *How do touch-screen monitors know where you're touching?*. http://computer.howstuffworks.com/question716.htm, Letzter Zugriff: 2013-02-02.

[CO01] connect.de *Resistiv: Touch durch Druck*. http://www.connect.de/ratgeber/resistiv-touch-durch-druck-377205.html, Letzter Zugriff: 2013-02-02

[SHW03] Schirmer, Höpfner *Smartphone Hardware Sensors*. http://www.uni-weimar.de/medien/wiki/images/Zeitmaschinen-smartphonesensors.pdf, Letzter Zugriff: 2013-02-02.

[MEMS1] St.J. Dixon-Warren *MOTION SENSING IN THE IPHONE 4: MEMS ACCELEROMETER*. http://www.memsjournal.com/2010/12/motion-sensing-in-the-iphone-4-mems-accelerometer.html, Letzter Zugriff: 2013-02-02

[ifix1] ifixit.com *iPhone 4 Gyroscope Teardown*. http://www.ifixit.com/Teardown/iPhone-4-Gyroscope-Teardown/3156/1s, Letzter Zugriff: 2013-02-02

[ifix2] ifixit.com *iPhone 5 Teardown*. http://www.ifixit.com/Teardown/iPhone+5+Teardown/10525/4, Letzter Zugriff: 2013-02-02

[MEMS2] St.J. Dixon-Warren *MOTION SENSING IN THE IPHONE 4: MEMS GYROSCOPE*. http://www.memsjournal.com/2011/01/motion-sensing-in-the-iphone-4-mems-gyroscope.html, Letzter Zugriff: 2013-02-02

[MEMS3] St.J. Dixon-Warren *MOTION SENSING IN THE IPHONE 4: ELECTRONIC COMPASS*. http://www.memsjournal.com/2011/02/motion-sensing-in-the-iphone-4-electronic-compass.html, Letzter Zugriff: 2013-02-02

[GPS01] NAVIaktiv.de *Grundlagen der GPS-Technik*. http://www.naviaktiv.de/index.php?cont=gpstechnik, Letzter Zugriff: 2013-02-02

[CB01] COMPUTERBILD.DE *Was ist A-GPS und welche Empfänger gibt es?*. http://www.computerbild.de/artikel/cb-Ratgeber-Navigation-Alles-ueber-Satellitennavigation-mit-GPS-3035514.html, Letzter Zugriff: 2013-02-02

[YT01] MARTINDATI *iPhone: Annäherungssensor*. http://www.youtube.com/watch?v=qcMMt5jOy7w, Letzter Zugriff: 2013-02-02

[AL01] ELEKTRONIKNET.DE *Umgebungslichtsensor*. http://www.elektroniknet.de/lexikon/?s=2&k=U&id=34416\&page=1, Letzter Zugriff: 2013-02-02

[IH01] INSIDE-HANDY.DE *Sensoren*. http://www.inside-handy.de/lexikon/sensoren, Letzter Zugriff: 2013-02-02

[IT01] IT WISSEN *MEMS microphone*. http://www.itwissen.info/definition/lexikon/MEMS-Mikrofon-MEMS-microphone.html, Letzter Zugriff: 2013-02-02

[IN01] INSIDE-HANDY.DE *iPhone 5: 3 Mikros, bessere Lautsprecher und Rauschunterdrückung*. http://www.iphone-news.org/2012/09/13/iphone-5-3-mikros-bessere-lautsprecher-und-rauschunterdruckung-42621/, Letzter Zugriff: 2013-02-02

[STM01] STMICROELECTRONICS *VB6850*. http://html.alldatasheet.com/html-pdf/231969/STMICROELECTRONICS/VB6850/1943/1/VB6850.html, Letzter Zugriff: 2013-02-02

[DI01] DIRECT INDUSTRY *Piezomotor für Autofokus-Objektive von Smartphone-Fotoapparaten*. http://www.directindustry.de/prod/new-scale-technologies/piezomotoren-fur-autofokus-objektiv-von-smartphone-fotoapparat-30035-657351.html, Letzter Zugriff: 2013-02-02

[pw08] PHYSICSWORLD.COM *8mm Piezo Motor / Linear Slide Ideal for Fiber & Optics Alignment, Autofocus*. http://physicsworld.com/cws/product/P000005991-8mm-piezo-motor-linear-slide-ideal-for-fiber-optics-alignment-autofocus, Letzter Zugriff: 2013-02-02

[PI01] PHYSIKINSTRUMENTE.COM *PILine Ultrasonic Piezomotor Systems - Working Principle*. http://www.physikinstrumente.com/en/products/piezo_motor/piline.php, Letzter Zugriff: 2013-02-02

[CW01] CHIPWORKS *Teardown of the Apple iPhone 4 Smart Phone*. http://www.chipworks.com/blog/recentteardowns/2010/06/23/silicon-teardown-of-the-apple-iphone-4-smart-phone/, Letzter Zugriff: 2013-02-02

[MN01] MEMSNET.ORG *What is MEMS*. https://www.memsnet.org/mems/what_is.html, Letzter Zugriff: 2013-02-02

[EK01] ELEKTRONIK-KOMPENDIUM.DE *MEMS - Micro-Electro-Mechanical Systems*. http://www.elektronik-kompendium.de/sites/bau/1503041.htm, Letzter Zugriff: 2013-02-02

Seminar Handybetriebssysteme

Android, Grundlagen und Programmierung

Lutz Malte Wagner
Fakultät II
Carl von Ossietzky Universität Oldenburg
Oldenburg, Deutschland
lutz.malte.wagner@uni-oldenburg.de

Dieses Paper soll einen Überblick über Googles Betriebssystem Android geben. Dabei gehe ich speziell auf die Technischen Grundlagen, Unterschiede der einzelnen Versionen, die Programmierung und von Android abgeleitete Betriebssysteme ein.

Android; Google; Open Handset Alliance; Linux; Bionic; C; Dalvik; Java; SDK; NDK; ADT; App Inventor;

I. Was ist Android

Android wurde von der Firma Android Inc entwickelt, welche schon sehr früh von Google aufgekauft wurde. Die späteren Versionen sind stark von den Richtlinien und Bestimmungen der Open Handset Alliance geprägt. Kein großes Wunder, da neben Firmen wie Sony, Samsung und LG auch Google dort vertreten ist. Mittlerweile steht Android bei über 75% Marktanteil und ist daher mit Abstand das Erfolgreichste Smartphone Betriebssystem. Aber was Machte Android so erfolgreich? [1]

Es besitzt einige Merkmale die es sowohl für Nutzer als auch für Hersteller und Programmierer schmackhaft machen. Google finanziert sich im Gegensatz zu den anderen Herstellern fast ausschließlich über Werbung und kann daher auf Lizenzkosten verzichten um die Verbreitung voran zu treiben. Diese Tatsache und das es eine Linux-Basis besitzt die es auf fast jeder Hardware lauffähig macht erleichtern natürlich die Arbeit der Hersteller. An der Java -unterstützung erfreuen sich die Programmierer und überfluten den „Google marketplace“ mit passenden Applikationen (oder kurz Apps) für nahezu jede Situation. Die große Auswahl an Smartphones mit Android als Betriebssystem und die fülle an Apps macht es wiederum zum Liebling der Nutzer. Damit werden wieder mehr Hersteller und Programmierer auf Android aufmerksam. Die so entstehende Spirale des Erfolgs konnte bisher noch kein anderes Betriebssystem durchbrechen.

II. Systemarchitektur

Android wird im Allgemeinen in 4 Ebenen unterteilt die auf einander Aufbauen.

- Wie bereits angesprochen basiert Android auf einem Linux-Kernel. Allerdings wird dieser Oberflächlich unterdrückt und nur über die Typischen Touchscreen und Tastatur eingaben angesteuert. Dies hat den einfachen Grund das ein Smartphone Betriebssystem mobil, handlich aber vor allem leicht und schnell zu bedienen sein muss. Ein Befehl im Kernel eintippen kann unter diesen Gesichtspunkten beispielsweise mit einem schnellen wisch über das Display nicht mithalten. Es gibt noch einige weitere drastische Unterschiede zu herkömmlichen Linux basierten Betriebssystemen. Neben veränderter Speicher und Grafikverwaltung nutzt Android beispielsweise keine GNU Elemente und beinhaltet eine selbst entwickelte C Bibliothek namens Bionic. Diese ist zwar von der normalen Linux Bibliothek glibc abgeleitet aber stark komprimiert, es fehlen Elemente wie die Unterstützung von wide charactern oder das Exception handling. Außerdem Interpretiert Android die uid's, eigentlich zur Identifizierung von Nutzern, um Apps zu identifizieren und die für Gruppen gedachten gid's um den Apps bestimmte Rechte zuzuweisen. Auch alle benötigten Treiber für die Hardware, also die Kamera, den Bildschirm, WiFi, Batterie und vielem mehr, befindet sich an dieser stelle. [2]
- Direkt danach kommen spezielle Grundbibliotheken für verschiedenste Bereiche und die Dalvik VM. Zu den Grundbibliotheken gehören beispielsweise das WebKit, zur Interpretation von den verschiedenen im Internet genutzten Sprachen, oder SQLlite zur Verwaltung von Datenbanken. Die Dalvik VM dient der Interpretation von Java Bytecode, diesen interpretiert sie nicht sofort, sondern wandelt ihn erst in „Dalvik executable code“ (kurz „dex“) um und interpretiert diesen dann. Bei der Art der Interpretation gibt es einige Grundlegende Unterschiede, so nutzt die Dalvik VM nicht wie Java typisch einen Kellerspeicher sondern greift auf Register zurück. Insgesamt nutzt die Dalvik VM weniger Arbeitsspeicher und benötigt weitaus weniger Prozessorleistung, da die mittel eines Smartphones in dieser Hinsicht eher beschränkt sind. Die Umwandlung zu Dalvik executable Code hat außerdem den Vorteil das nicht direkt mit Java Bytecode gearbeitet wird und somit nach Meinung von Google auch keine Lizenzkosten an Oracle zu zahlen sind. Diese versuchten Google gerichtlich zu

einer Zahlung zu zwingen. Im Mai 2012 wurde Google entschieden das Google keine Patente verletzt hat. [3]

- In der dritten Ebene befindet sich das so genannte „Application Framework“, welches sich allgemein um die Verteilung von Ressourcen kümmert. Hierbei gibt es für nahezu jede Ressource auch einen eigenen Manager, zum Beispiel den Window-Manager für einzelne Oberflächen, den Activity Manager der die Prozesse regelt und überwacht. Grob umschrieben passt diese Ebene also auf das jede Anwendung die Ressourcen bekommt die es braucht und erhalten darf.
- Die oberste und letzte Schicht stellen die Oberflächen die ein Nutzer letztendlich zu Gesicht bekommt und die Logik der dazugehörigen Apps. Hier befinden sich der Startbildschirm, die verschiedenen Menüs, der Webbrowser, Programme zum nutzen der Kamera oder des GPS und alle Programme die von dritten geschrieben und auf dem Smartphone installiert werden. [4]

III. Versionen

Android gibt es in 32 verschiedenen Versionen, von denen nur die ersten beiden Versionen, 1.0 und 1.1, mittlerweile nicht mehr unterstützt werden. Dies ist vor allem für die Programmierung von Apps wichtig, da mit den meisten dieser Versionen auch neue APIs erschienen sind. Da viele Hersteller ihr Smartphones nur langsam beziehungsweise gar nicht updaten ist auch ein großer Teil von ihnen noch mit alten Versionen unterwegs. Auch die unterscheide dieser Versionen sind teilweise gravierend, ob mit Version 1.5 („Cupcake“) der Drehbare Bildschirm und die Bildschirmtastatur, mit 2.1 („Eclair“) html5 oder mit 2.2 („Froyo“) der Zugang zu den SD-karten eingeführt wurden, es ändert sich fast alles und das Verhältnismäßig schnell.

Auch ein wichtiger Aspekt ist das mit Version 3.0 („Honeycumb“) erstmals auch auf Tablets eingegangen wird. Dies zeigt sich beispielsweise in der Unterstützung von höheren Auflösungen oder mehr Arbeitsspeicher. Erst mit Version 4.0 („Ice Cream Sandwich“) werden die damit entstehenden Probleme gelöst und ein Betriebssystem für Smartphones und Tablets gleichermaßen entsteht.

Als letzte große Änderung kommt mit Version 4.2 („Jellybean“) der Mehrbenutzerbetrieb, zumindest für Tablets.

Außerdem wurde in größeren aber regelmäßigen abständen der Linux-Kernel erneuert. Dies ergibt sich da mit diesen Linux Versionen Verbesserungen kamen die auch für ein Smartphone Betriebssystem von wert sind, wie eine besseres Management des Arbeitsspeichers oder dergleichen. Allerdings ist ein solches Update auch Zeitaufwendig, da die einzelnen Unterschiede zwischen Standard Linux und Android Linux auch neu angepasst werden mussten. Hinzu kommt das Android ein paar dieser Änderungen bereits im Vorfeld implementiert hatte und somit eine Umstellung nicht von Nöten war. [5]

IV. Portierungen und Derivate

Da die Quellen von Android frei zugänglich sind und keine rechtlichen Gegebenheiten eine Portierung des Betriebssystems verbieten gibt es mittlerweile einige mehr oder weniger vollständige Betriebssysteme auf Android Basis für die verschiedensten Systeme. Ein paar dieser Projekte wurde von Google oder auch nur einzelnen Google unterstützt. Die meisten befinden sich noch im Alpha Stadium und sind noch nicht ausgereift. Allerdings wird an Portierungen auf jedes gängige Smartphone Betriebssystem gearbeitet. XDAndroid für Windows Mobile [6], AndroidPlayer für Blackberry [7], Anrdomnia für Omnia [8] und viele mehr.

Einiges an Aufsehen erregte das Aliyun OS der Alibaba Group, dieses wurde als Konkurrent von Android auf dem Asiatischen Markt entwickelt. Es besitzt ebenfalls ein Linux-Kernel und verwendet einige Android Frameworks. Außerdem finden sich in seinem App store einige Android Apps, unter anderem auch Google eigene Dienste wie Google translate. Die Alibaba Group verweist darauf das Aliyun auf Cloud Anwendungen setzt und sich damit Grundlegend von Android unterscheidet während Google es als „inkompatible Android Abspaltung“ betitelt. Da Aliyun zwar unter der „GNU General Public License“ veröffentlicht wurde aber der Quellcode trotzdem nicht zugängig gemacht wurde, ist unbekannt wieweit Aliyun wirklich auf Android basiert. Daher ging Google so weit das sie Acer mit Beendigung der Partnerschaft drohten als diese ein Smartphone mit Aliyun veröffentlichen wollten. Da Acer zu über 90% in ihren Systemen Android verwendet zogen diese die Kooperation mit der der Alibaba Group daraufhin zurück. [9]

Am weitesten verbreitet ist der CyanogenMod. Seine Entwicklung wurde relativ früh von einer großen Community begonnen und erhielt schnell mithilfe von Google Mitarbeitern. Eingebremst wurde das Projekt als es seitens Google eine Abmahnung gab, da nicht lizenzfreie Apps, wie Googlemail, verwendet wurden. Es wurde sich darauf geeinigt das Diese nicht mehr im Grundpaket enthalten sein dürfen, aber bei bedarf während der Installation aus dem bestehenden Android Betriebssystem herauskopiert werden können. Das Betriebssystem bietet einige Möglichkeiten zum übertakten und ist nach eigenen Angaben von Haus aus schon schneller als das Basis Android. Außerdem beinhaltet es für einige Smartphones schon alle Inhalte der aktuellsten Version, unterstützt alle Apps und ist bis auf kleinere Probleme Fehlerfrei. [10]

Aber wo liegt das Problem bei den Portierungen? Ein Linux basierendes Betriebssystem ist normalerweise nahezu überall lauffähig und auch Java ist Plattformunabhängig. Das größte Hindernis ist die Optimierung in Android, hier wurde an vielen Stellen zugunsten der Performance auf Flexibilität verzichtet um mit den begrenzten Ressourcen eines durchschnittlichen Smartphones immer noch eine gute Leistung erzielen zu können. So ist der Register basierte Speicher der Dalvik VM nicht mit jedem System ohne Anpassungen kompatibel und besonders schwierig wird es wenn das Gerät keine ARM-Architektur besitzt, in diesen

Fällen kommt es meistens dazu das die meisten Grundelemente ausgetauscht werden müssen, da Android auf diese Architektur ausgelegt ist. Es wurden zwar später noch die Unterstützung von ARM-NEON sowie X86 und MIPS nachgereicht, allerdings gibt es auch hiermit teilweise noch Probleme. [11]

V. Programmierung

Der Übliche weg zur Entwicklung von Applikationen für Android ist das Android SDK für Java. Hierüber erstellte Applikationen werden wie System eigene Anwendungen über die Dalvik VM interpretiert und ausgeführt. Seit Android 1.5 gibt es zudem eine Möglichkeit nativem Code zu schreiben, welcher an der Dalvik VM ignoriert und über die Bionic C Bibliotheken bearbeitet wird. Somit ist es möglich eigene Applikationen beispielsweise in C oder C++ zu erstellen. Für Neulinge gibt es außerdem eine Art Baukasten der kostenlos nutzbar ist.

Zu einem Projekt gehören immer mehrere Attribute die festgelegt werden müssen, am wichtigsten ist natürlich der eigentliche Name der App. Außerdem muss es zu jeder App ein Projekt geben, gibt man für mehrere Apps das selbe Projekt an kann Android so erkennen das sie direkt zusammengehören. Des weiteren benötigt es einen Paketnamen, unter diesem wird die App später abgespeichert. Der Paketname muss als einziges Eindeutig sein, ist er es nicht kann es zu Überschneidungen kommen. Alle gewählten Voreinstellungen stehen später in der Projekt eigenen Manifest Datei, können also jederzeit wieder geändert werden.

A. Applikationen in Java

Die Entwicklung von Anwendungen mit Hilfe von Java wird über das Android SDK ermöglicht. Zu diesem gehört ein umfangreiches PlugIn für die Java Entwicklungsumgebung eclipse genannt NDK (steht für Android Development Tool). Es enthält unter anderem einen eigenen GUI-builder.

Schon beim erstellen eines eigenen Projektes fällt auf das Google sich einige Gedanken über Versionsübergreifende Apps gemacht hat und dem Entwickler hier einige Möglichkeiten bietet. Neben einer natürlich benötigten Version in der Kompiliert werden soll kann beispielsweise auch eine Version angegeben werden die mindestens unterstützt wird. Dies ist möglich da die Dalvik VM während des Vorkompilieren bei unbekanntem Code nicht sofort einen Fehler ausgibt sondern dies erst direkt bei der Ausführung betreffender Zeile geschieht. So können durch geschicktes abfragen bestimmte inkompatible stellen umgangen werden und somit eine App sogar in geringeren Versionen genutzt werden. Auch wenn hierbei dann die Funktionalität eventuell eingeschränkt werden muss. Im verlauf des Anlegens gibt es einige weitere Optionen die den Einstieg erleichtern sollen. Hier kann beispielsweise eine Vorlage für eine erste Oberfläche ausgewählt werden.

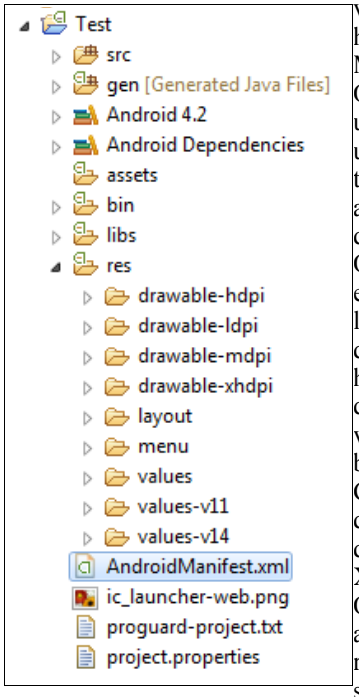
Es werden nicht alle aus einer normalen JavaRE bekannten Befehle unterstützt, aber mit folgenden Paketen sind die wichtigsten Bereiche übernommen worden:

- java.io
- java.lang
- java.math
- java.net
- java.nio
- java.security
- java.sql
- java.text
- java.util
- javax.crypto
- javax.net
- javax.security
- javax.sound
- javax.sql
- javax.xml.parsers
- org.w3c.dom (ohne Unterpakete)
- org.xml.sax

Zudem werden ein paar Fremdbibliotheken eingebunden:

- org.apache.commons.codec
- org.apache.commons.httpclient
- org.bluez
- java.net

Für einige weitere Bereiche, wie Oberflächen oder der Umgang mit XML, wurden von Google eigene Bibliotheken entwickelt. [12]



Solange man sich an die von dem Android SDK vorgegebene Datenstruktur hält gibt es einige Möglichkeiten den Quellcode zu vereinfachen und vor allem aussehen und Logik von einander zu trennen. Dies wird vor allem dadurch ermöglicht das sich die Android Oberflächen als XML erstellen und abspeichern lassen, später können diese dann aus dem Quellcode heraus aufgerufen und mit der nötigen Logik versehen werden. Auch der bereits angesprochene GUI-builder arbeitet nach diesem Prinzip und erstellt dynamisch eine passende XML Datei. Diese XML Oberflächen müssen sich allerdings im Ordner res/layout befinden damit sie später von Android gefunden werden.

Auch in dem Ordner res befinden sich der Unterordner drawable, in welchen die Bilder gespeichert werden sollten. Zusätzlich kann man hier über verschiedene Endungen verschiedene Versionen in unterschiedlicher Auflösung

speichern, aus welchen Android selbstständig das am besten passende heraussucht, zum Beispiel ist drawable-ldpi für eher geringere Auflösungen zuständig (ldpi steht für low dpi) während drawable-xhdpi (xdhpi für extreme high dpi) für die viel höheren Auflösungen von Tablet PCs gedacht ist. Dazwischen befinden sich noch mdpi (medium dpi) und hdpi (high dpi). In der Projektspezifischen Manifest Datei lässt sich hierzu auch einstellen das nur bestimmte Auflösungen oder Bildschirmgrößen unterstützt werden. Besitzt das Ausführende System eine andere Auflösung kann die App dann nicht gestartet werden oder meist nicht mal heruntergeladen werden. [13] [14]

Ähnliches gilt für die Lokalisierung. Sie wird über den Unterordner values realisiert, hier werden xml Dateien mit verschiedenen String werten angelegt. Ähnlich der Lokalisierung über propertie Dateien im normalen JavaRE können hier über Endungen verschiedene Sprachen und Länder angesteuert werden. Wird sie nicht gefunden wird automatisch die nächst höhere Ebene genutzt. Greift eine App beispielsweise innerhalb Deutschlands auf eine strings.xml zu, so wird intern zunächst nach einer strings-de-DE.xml, die für Deutschland zuständig ist. Sollte diese nicht gefunden werden so wird die für den deutschsprachigen Raum genutzte strings-de.xml gesucht. Wird auch diese nicht gefunden gilt automatisch das was in der allgemeinen strings.xml angegeben wurde. Da mit neueren Version dieses verbessert und erweitert wurde können außerdem values-v11 und values-v14 verwendet werden um diese Funktionen zu nutzen und trotzdem abwärtskompatibel zu bleiben.

In res befindet sich außerdem der Ordner menu, in diesen können Oberflächen gespeichert werden die angezeigt werden sollen wenn auf dem Smartphone die Menütaste gedrückt wird. Auch hierfür gibt es einen eigenen GUI-builder. Dieser bietet alle Elemente die zum erstellen eines Menüs gebraucht werden, sogar einzelne einfache Funktionen lassen sich hier direkt erstellen.

Der Ordner assets ist für Datenströme zuständig, hierfür stellt Android einige Methoden zur Generierung von Input- beziehungsweise OutputStreams zur Verfügung. Er ist allerdings optional und befindet sich daher nicht im Ordner res sondern steht auf der selben ebene.

Das wahrscheinlich wichtigste Konzept sind die so genannten Intents. Über sie gibt es die Möglichkeit neue Oberflächen zu öffnen, spezielle Funktionen zu starten, eine App selber bei bestimmten aufrufen starten zu lassen und vielem mehr. Im Großen und Ganzen kann man sagen sie dienen der Kommunikation innerhalb einer Applikation aber auch unter den verschiedenen Applikationen. Beispielsweise gibt es Intents denen man die Adresse einer Website mitgibt, nun ist es egal welche Applikation als Standard Webbrowser ausgewählt wurde, beziehungsweise eventuell auch keiner, Android kümmert sich darum und sorgt dafür das sie angezeigt wird. Ähnlich verhält es sich mit Telefonnummer, SMS, Emails und vielem anderen. Der Entwickler sagt was gemacht werden soll und Android entscheidet je nach Einstellungen welche App sich darum kümmert.

Das Gegenstück zu den Intents sind die IntentFilter, über diese kann das Programm auf Intents reagieren, so kann zum Beispiel eine Applikationen immer gestartet werden wenn die Website http://www.uni-oldenburg.de aufgerufen wird. Somit ist es jedem Entwickler möglich beispielsweise einen eigenen Webbrowser, ein eigenes Mail Programm oder ähnliches frei nach belieben zu erstellen ohne sich darum kümmern zu müssen wer diese Apps später nutzt und ob es eventuell Konkurrenzprodukte, wie andere Webbrowser, gibt.

Hiermit kommen aber auch große Risiken im Bereich der Datensicherheit ins Spiel. Wenn es einer App erlaubt ist auf Email, Telefonnummern und ähnliches zuzugreifen wird es Hackern, oder Betrügern im Allgemeinen, besonders leicht gemacht an benötigte Daten zu gelangen. Beispielsweise kann so jede Telefonnummer die gewählt wird abgefangen und gespeichert werden. An jede E-Mail können zwischen der absende Aufforderung des Nutzers und dem tatsächlichen absenden noch beliebig viele Daten angehängt oder verändert werden, ohne das dieser etwas davon mitbekommt. Um dies zu verhindern muss der zugriff auf bestimmte Intents, wie gewählte Telefonnummern, erst vom Nutzer erlaubt werden. Geschieht dies nicht werden entsprechende Daten niemals an die App gesendet und sind somit vor Fremdzugriffen gesichert. Diese nachfrage geschieht während der Installation. [15]

B. Applikationen in Nativem Code

Android unterstützt seit Version 1.5 die Programmierung in nativem Code. Dieser ist naturgemäß schneller in der Ausführung, da er nicht erst noch erneut interpretiert werden muss, wie es beispielsweise für Javas Bytecode der Fall ist. Zur Entwicklung mit nativem Code wurde von Google das NDK (kurz für Native Development Kid) zur Verfügung gestellt. Allerdings werden auch hier aufgrund der Einschränkungen der in Android verwendeten Bionic C Bibliothek nicht alle Befehle unterstützt, was gerade durch Fehlendes Exception handling schnell zu einem sehr unübersichtlichen Quellcode führen kann. Aus diesen Gründen ist es eher unüblich das ganze Apps in nativem Code verfasst werden, die Möglichkeit bestimmte Schnittstellen in nativem Code zu schreiben und aus Java Code darauf zurückzugreifen ist allerdings auch gegeben und wird häufiger genutzt. Gerade im Bereich der mittlerweile zahlreichen Spiele ist hierdurch ein großer und notwendiger Geschwindigkeitsvorteil vorhanden. Für die meisten anderen Anwendungen bringt es allerdings keine Vorteile und schränkt einen durch die Fehlenden Funktionen sogar noch weiter ein, was Google dazu brachte bei dem Download des NDK bereits darauf Hinzuweisen das dieses in den meisten fällen keinen Vorteil bringt. [16] [17]

C. Applikationen über den App Inventor

Der App Inventor ist eine Art Baukasten für Android Applikationen. Entwickelt wurde er von Google als Teil von Google Labs, als dieses eingestellt wurde übernahm es das MIT, da eine Grundlegende Bibliothek dort erstellt wurde, und stellte es nach kurzer Zeit erneut für die Öffentlichkeit zu Verfügung. Der App Inventor ist zunächst über eine Website zu erreichen, für das erstellen der Programmlogik und zum testen müssen allerdings Programme

heruntergeladen werden. Die Entwicklung startet mit dem designen der Grafischen Oberfläche innerhalb der Website, hierfür steht eine stark abgespeckte Version des GUI-builders aus dem SDK zur Verfügung. Diesem Fehlen beispielsweise fast alle Panelarten, sowie die spezielleren Einstellungsmöglichkeiten für Textfelder, Buttons und ähnlichem. Nachdem die Oberflächen erstellt sind kann durch das aneinander und ineinander Schieben von Puzzle teilen ähnelnden Blöcken die Logik erstellt werden. Das Aussehen dieser Oberflächen erinnert stark an die Java Lernplattform Scratch, sie basieren auch beide auf der selben Java Bibliothek, Open-Blocks. Dieses wurde von Ricarose Roque als Masterarbeit am MIT veröffentlicht und später dort im Rahmen des „Scheller Teacher Education Program“ (kurz STEP) unter der MIT-License veröffentlicht. Zum Testen der Apps steht außerdem ein Emulator zur Verfügung. [17] [18]

D. Die Veröffentlichung

Will man seine fertigen Apps im Google marketplace veröffentlichen benötigt man einen Entwickler Account dessen Erstellung einmalig 25$ kostet, zusätzlich behält Google 30% der einnahmen für sich. Vorausgesetzt wird das jede App einen eindeutigen Pfad besitzt da es ansonsten zu Überschneidungen auf dem Endgerät kommen kann. Ist ein Pfad bereits vergeben muss man diesen ändern oder sich eine andere Stelle suchen um die App zu veröffentlichen. Da es möglich ist Programme über den PC auf dem Smartphone zu installieren gibt es mittlerweile verschiedene Foren in denen man seine Programme kostenlos zur Verfügung stellen kann. Hierdurch spart man sich die kosten für einen Entwickler Account und muss nicht zwingend einen eindeutigen Pfad angeben, allerdings erreicht man hier auch nicht die Masse die eine App in Googles marketplace erreichen kann und man muss damit rechnen das der Pfad eventuell schon vergeben ist. [19]

VI. Fazit

Android ist als Googles Smartphone-Betriebssystem bekannt geworden, mittlerweile kann man es weder als reines Smartphone-Betriebssystem betrachten noch sollte Google als alleiniger Entwickler genannt werden. Es wird von vielen Weiterentwickelt und das für fast jede Plattform. Es ist ein Kompromiss zwischen Leistung und Flexibilität der die Kunden überzeugt und den Herstellern trotzdem die Möglichkeit gibt es fast überall zu verwenden. Programmierern bietet es die Möglichkeiten fast alles damit anzustellen und es im Google marketplace kostenlos jedem zur Verfügung zu stellen, was wieder neue Kunden anlockt und somit das Betriebssystem noch lukrativer für die Hersteller und Programmierer macht. Eine Spirale die bisher noch nichts unterbrechen konnte.

Für Entwickler ist das sicherlich erfreulich, da somit erstellte Apps wahrscheinlich noch Jahre lang nutzbar bleiben. Androids Dalvik Java bietet viele Vorteile gegenüber dem Herkömmlichen Java, wenn man sich an die Unterschiede erst mal gewöhnt hat sind diese auch durchaus Praktisch und überwiegen für mich die Nachteile.

Es bleibt abzuwarten was mit Android 5.0 und den darauf folgenden Versionen für Veränderungen kommen. Vielleicht werden ein paar Ideen auch in die normalen Java Bibliotheken übernommen, hierfür müssten allerdings erst die Streitigkeiten zwischen Oracle und Google enden.

[1] http://de.statista.com/themen/581/smartphone/infografik/291/weltweiter-marktanteil-der-smartphone-betriebssysteme/ (03. Februar 2013)

[2] http://www.zdnet.de/41553061/android-architektur-wieviel-linux-steckt-in-googles-os/ (03. Februar 2013)

[3] http://www.zdnet.com/blog/btl/jury-clears-google-of-infringing-on-oracle-patents/77897 (03. Februar 2013)

[4] http://www.pcwelt.de/ratgeber/Smartphone-Grundlagen-Technik-erklaert-Google-Android-Architektur-im-Detail-1005294.html (03. Februar 2013)

[5] http://de.wikipedia.org/wiki/Liste_von_Android-Versionen (03. Februar 2013)

[6] http://xdandroid.com (03. Februar 2013)

[7] http://developer.blackberry.com/android/ (03. Februar 2013)

[8] http://andromnia.sourceforge.net/ (03. Februar 2013)

[9] http://www.zdnet.com/is-aliyun-os-really-linux-android-a-rip-off-of-both-7000004318/ (03. Februar 2013)

[10] http://www.cyanogenmod.org/ (03. Februar 2013)

[11] http://webmagazin.de/einfuehrung-portierung-android-37649.html (03. Februar 2013)

[12] http://www.zdnet.com/blog/burnette/java-vs-android-apis/504 (03. Februar 2013)

[13] http://developer.android.com/guide/practices/screens_support.html (03. Februar 2013)

[14] http://developer.android.com/guide/practices/screens-distribution.html (03. Februar 2013)

[15] http://developer.android.com/guide/components/intents-filters.html (03. Februar 2013)

[16] http://developer.android.com/tools/sdk/ndk/index.html (03. Februar 2013)

[17] http://web.mit.edu/newsoffice/2010/android-abelson-0819.html (03. Februar 2013)

[18] http://appinventor.mit.edu/ (03. Februar 2013)

[19] http://www.heise.de/ct/hotline/FAQ-Eigene-Android-Apps-1155583.html (03. Februar 2013)

Handybetriebsystem: iOS

Ausarbeit zum Proseminar Handybetriebssysteme

Alexander Söker
Carl von Ossietzky Universität Oldenburg
Fakultät II - Informatik
Systemsoftware und verteilte Systeme
E-Mail: alexander.soeker@uni-oldenburg.de

Zusammenfassung - Im Folgenden wird eine Übersicht über das Handybetriebssystem iOS aus dem Hause Apple gegeben. Dabei sollen nach einer kurzen Motivation erst die allgemeinen Eigenschaften aufgeführt werden. Danach werden die unterschiedlichen Ebenen der Software aufgezeigt. Des Weiteren werden neben der Speicher- und Dateiverwaltung, auch noch die Grundarchitektur einer App für das iOS beschrieben. Abschließend soll ein Ausblick in Kombination mit einer Kritik stehen.

I. Einleitung

Die folgende Arbeit wurde im Rahmen des Proseminars "Handybetriebssysteme", besucht im Wintersemester 2012/13, erstellt. Es sollen die grundlegenden Eigenschaften und Besonderheiten des Handybetriebssystems iOS aus dem Hause Apple aufgezeigt werden. Neben den unterschiedlichen Abstraktionsebenen und dessen Aufgaben bzw. Möglichkeiten, soll auch die Grundstruktur und einige Desing Patterns für die Entwicklung von Apps(Kurzform: application = Anwendung) im Bereich des iOS genannt werden. Nachdem auf die Art der Speicherverwaltung und Archivierung von Daten eingegangen wurde, soll im weiteren Verlauf dann die Grundstruktur einer "handelsüblichen" App, wie sie im AppStore zur Verfügung steht, erklärt und betrachtet werden. Abschließend wird versucht, einen kritischen Ausblick auf die Weiterentwicklung im Bereich der APPs und des iOS an sich zu gegeben.

II. Motiviation

Die Bedeutung und Nutzung von Handys und Smartphones ist in den letzten Jahren exorbitant angestiegen. Gerade im Bereich der mobilen Nutzung des Internets bieten Smartphones eine optimale Arbeitsoberfläche[1]. Die Frage die sich nun stellt, ist, wie eben die genannten Smartphones grundsätzlich funktionieren. Dabei ist die Rede nicht von der speziellen Hardware der verschiedenen Geräte, sondern von den Betriebssystemen, die die Basis für ein erfolgreiches Smartphone darstellen. Oftmals sind die Fragen, die sich bei einem Kauf stellen, die nach einem bestimmten Typ von Smartphone oder einer bestimmten Marke. Jedoch sollte die Frage nach dem zu wählenden Betriebssystem weiter in den Vordergrund rücken. Bei einigen Modellen ist man bereits durch den Kauf schon an ein festes Betriebssystem gebunden. Das, an dieser Stelle beschriebene iOS-Betriebssystem, ist ein solches. So gibt es für die unterschiedlichen mobilen Apple-Produkte das hauseigene iOS vorinstalliert, dazu geliefert. Da es sich bei diesem Betriebssystem um ein geschlossenes Betirbessystem handelt, sind Modifikationen jeglicher Art nur schwer durchführbar.

Im Folgenden soll nun untersucht werden, was die Kernmerkmale und Besonderheiten des iOS-Betriebssystems sind und warum Apple mit diesem Betriebssystem der Marktführer im Bereich der Smartphones ist [1]. Aber auch etwaige Kritikpunkte an den Geschäftsmodellen rund um das iOS sollen beleuchtet werden.

III. Allgemeine Eigenschaften

Das Betriebssystem iOS basiert grundlegend auf dem Apple Betriebssystem Mac-OS-X. Erstmals wurde das iOS im iPhone 1 im Jahre 2007 genutzt, damals noch unter dem Namen iPhone OS. Es ist auf die spezielle ARM-Prozessorarchitektur angepasst und biete somit eine optimale Zusammenarbeit zwischen Hardware und Software an. In den folgenden Versionen wurden weitestgehend Updates vorhandener Funktionalitäten eingespeist und kleinere Funktionen (bspw. GoogleStreetView) hinzugefügt. Mit erscheinen des iPad im Jahre 2009, mussten im Betriebssystem mehrere Anpassungen vollzogen werden, damit es kompatibel zu einem größeren Display war. Erst mit der Version 4 im Jahre 2010 bekam das Betriebssystem seinen Namen iOS. Der nächste Meilenstein sollte im Jahr 2011 das iOS 5 darstellen, in dem die Funktionen der iCloud (ein Online-Speicherservice von Apple) und Siri (Spracherkennungssoftware) integriert wurden. Letztere kann man nur mit einem iPhone4s oder einem iPhone 5 nutzen.

In der aktuellen Version 6 wurden, wieder auf Grund eines neuen Endgerätes, Anpassungen vorgenommen. Mit dem Release des iPhone 5 wurden zusätzlich noch die Social-Networks Facebook und Twitter direkt in die Basisfunktionen eingebunden. Außerdem ersetzte die Apple Map, die bislang genutzte Google Map, als Standard Kartenservice.

Des Weiteren kann davon ausgegangen werden, dass es sich bei dem iOS um eines der weltweit am meisten eingesetzten Handybetriebssysteme handelt. So wurden bis zum Jahr 2011 ca. 50 Mio. iPhones, 35 Mio. iPods und 3 Mio. iPads mit iOS ausgestattet, verkauft. Auch dem eigens für iOS eingeführten AppStore (2008) folgten sämtliche andere Hersteller mit Ablegern für ihr Betriebssystem. Das

iPhone mit iOS revolutionierte den Markt für Smartphones und sollte schnell als eine Art Statussymbol für den Nutzer, sowie als Vorbild für andere Hersteller gelten[4].

A. Grundfunktionen

Das Betriebssystem ist auf den Apple Endgeräten vorinstalliert und kann auch nicht wieder vollständig entfernt werden. Es lässt sich also festhalten, dass es sich im Grunde um eine Pflicht handelt, das iOS auf Apple Produkten (iPhone, iPod und iPad) zu nutzen.

Um nun zu den Grundfunktionen des Betriebssystems zu kommen, lässt sich feststellen, dass vor allem die Basisapps aus dem Hause Apple die grundlegenden Bedürfnisse eines Nutzers stillen. So werden neben den Standardhandyfunktionen, der Telefon-, Adressverwaltungs- und SMS-Funktion weitere hilfreiche Anwendungen zur Verfügung gestellt. Eine Kartenapp, die Möglichkeit der Mail- und Kalendereinrichtung und natürlich der Zugang zum Einstellungsmenü sowie zu iTunes und zum AppStore stehen zur Verfügung. Auch das Lesen von eBooks ist seit der Version 6 integriert. Außerdem existiert eine kleine Anzahl von hilfreichen Apps aus dem Hause Apple. Hierbei handelt es sich beispielsweise um einen Taschenrechner oder einen Notizbock.

Im Bereich des Einstellungsmenüs befinden sich eine große Zahl an Optionen, die es möglich machen das System nach seinen eigenen Bedürfnissen anzupassen (beispielsweise Lautstärke, Helligkeit, Netzanbieter des Endgerätes).

Seit der Version 6 gibt es auch die Möglichkeit per „Fingerdrop“ am oberen Bildrand ein Nachrichtencenter aufzurufen und so die neusten Meldungen der unterschiedlichen Apps auf einen Blick zu betrachten[2].

B. Unterstützung von Peripherie

Das iOS unterstützt neben dem externen Ladegerät noch viele weitere externe Geräte. So kann mit Hilfe eines Adapters der Bildschirminhalt auch auf externen Geräten (VGA / HDMI) angezeigt werden. Das Anschließen von externen Audiowiedergabegeräten ist möglich. Des Weiteren ist auch eine Kombination aus Ladegerät und Wiedergabegerät (Docking-Station) ohne Probleme nutzbar.

Häufig werden geraden im Bereich der Peripherie-Geräte die Produkte aus dem Hause Apple besser unterstützt, als jene von Drittherstellern. Das Anschließen von anderen Peripheriegeräten, wie beispielsweise kleinen Joysticks oder Tastaturen geschieht häufig auch über die unterstützte Bluetooth-Schnittstelle. So können auch Freisprecheinrichtungen oder Bluetooth-Anlagen angesteuert werden. Auch im Bereich der Automobilbranche wird sich diese Schnittstelle häufig zu nutzen gemacht. Ein passendes Beispiels hierfür ist der Automobilhersteller Opel, der es mit Hilfe einer iOS-App erlaubt via Bluetooth, Fahrzeugdaten auf dem iPhone anzuzeigen (bspw. Kühlwassertemperatur usw.) [3].

C. Nutzung der Hardware

Das Iphone 5 ist mit einer großen Anahl an Sensoren und Aktoren ausgestattet. Da das iOS speziell auf dieses Gerät abgestimmt ist, kann es diese auch optimal nutzen. Zu den Sensoren und Aktoren, sowie der unterstützen Hardware, gehören:

- **Mikrophon**, als Spracheingabe bei Telefonaten, Aufnahme von Memos und Bedienung der Sprachsteuerung Siri
- **Lautsprecher**, als Freisprecheinrichtung und zur Wiedergabe von Musik oder Tönen.
- **Kamera** (Rückseite/Vorderseite), zur Nutzung der Videotelefonie (Facetime) und Foto-und Videokamera.
- **Vibrationsgenerator**, als Signal für eine Nachricht oder Anruf.
- **GPS-Sender/Empfänger**, zum Ermitteln der genauen Position und damit als Grundlage für die Navigation, sowie zur Nutzung der „Where-is-my-iPhone“-Funktion (Suchfunktion, die die Position des iPhones über das Internet bestimmt)
- **WLAN**, zur Nutzung von WLAN-Netzen in der Umgebung.
- **Bluetooth**, als Schnittstelle für diverse externe Geräte (Musikwiedergabe, Datenübermittlung).
- **Beschleunigungssensor** und **Gyroscope**, als eine intuitive Bedienfunktion, die beispielsweise durch Schütteln des Endgerätes gewisse Funktionen auslöst.
- **Digitaler Kompass**, zur Orientierung in Map-Services.
- **Umgebungslicht-Sensor**, zur automatischen Regulierung der Bildschirmhelligkeit.
- **Annäherungssensor**, der es ermöglicht, dass Display bei einem Telefonat auszuschalten, solange es in Ohrnähe ist, damit Fehleingaben verhindert werden können.

Alle genannten Hardwarebausteine werden von dem Betriebssystem iOS in seiner neusten Version 6 unterstützt und den unterschiedlichen Apps zur Verfügung gestellt[4].

D. Besonderheiten

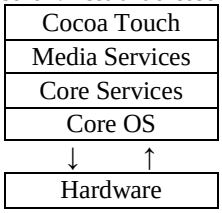
Die Grundarchitektur des iOS Betriebssystems ist so ausgelegt, dass lediglich eine App zur selben Zeit ausgeführt werden kann. Somit wird der aktiven Applikation das gesamte Fenster bzw. Display zur Verfügung gestellt und zwei Apps können nicht nebeneinander angezeigt werden. Es ist lediglich möglich Apps im Hintergrund weiterlaufen zu lassen. Außerdem wird das iPhone bzw. iPad mit dem iOS lediglich mit nur einer richtigen Taste bedient und kann somit komplett nur über den eingebauten Touchscreen bedient werden.

Zu den Bersonderheiten im Bereich der Speicher- und Dateiverwaltung wird in den Punkten VI und VII eingegangen.

IV. EBENEN DES IOS

Das Betriebssystem iOS kann grob in vier Abstraktionsschichten eingeteilt werden, die alle aufeinander aufbauen, um die Hardware des Endgerätes anzusprechen. Umso weiter eine Schicht von der Hardwareebene entfernt ist, umso weiter wurde sie abstrahiert. Jede der vier Ebenen stellt seine eigenen Eigenschaften und Funktionen zur Verfügung. So werden durch diese vier Abstraktionsebenen alle notwendigen Funktionen abgedeckt und stehen dem iOS-App Entwickler zur Verfügung.

Tabelle 1: Abstraktionsebenen



Im Folgenden werden nun die vier Abstraktionsschichten, siehe Tabelle I, genauer erklärt und beschrieben. Das Hauptaugenmerk liegt dabei auf der Basis und somit hardwarenahsten Schicht: dem Core OS.

A. Core OS

Das Core OS ist die niedrigste Abstraktionsschicht und enthält neben den grundlegenden Funktionen keine anderen Funktionen als die höher liegenden Schichten. Diese Schicht wird nur selten von Programmierer genutzt, jedoch sehr häufig von den anderen Abstraktionsschichten.

Zu den Kernframeworks des Core OS zählen:

Das **Accelerate Framework**: Dieses Framework enthält die Interfaces für die DSP-Bearbeitung (Verarbeiten von einzelnen digitalen Signalen), lineare Algebra und grundlegende Fertigkeiten zur Berechnung von Bildern (image-processing-calculations).

Das **Core Bluetooth Framework**: Wie der Name schon vermuten lässt, handelt es sich um das Framework, das dem Entwickler erlaubt die eingebaute Bluetooth-Hardware ordnungsgemäß zu nutzen. So werden die Standartoperationen, wie Verbinden, Trennen, Schreiben, Lesen und Finden zur Verfügung gestellt.

Das **ExternalAccessory Framework**: Durch dieses Framework wird die Möglichkeit gegeben, externe Geräte anzusprechen bzw. sich Informationen über sie zu beschaffen. Bei externen Geräten ist hier in erster Linie die Rede von Geräten, die über den StandardiPhone Anschluss (30-Pin) oder kabellos via Bluetooth angeschlossen sind.

Das **General Security Services Framework**: Dieses Framework unterstützt die Standard Funktionen bezüglich der Sicherheit. So werden die standardisierten Fälle RFC 2743 und RFC 4401 abgedeckt, sowie einige weitere grundlegende Sicherheitsoptionen zur Verfügung gestellt. Bei den RFC (Request for Comments) handelt es sich um formalisierte Fälle im Bereich des Internets und der Softwareentwicklung.

Das **Security Framework**: Zusätzlich zu dem General Security Services Framework arbeitet dieses Framework im Bereich des Zertifikatsmanagement und Trust-Policies. So kann mit diesem Framework eine Reihe kryptographischen Schlüsseln gespeichert und verwaltet werden. Dies ist gerade im Bezuge auf benutzerkontenbasierten Apps ein sehr hilfreiches Mittel. Auch die sichere Kommunikation zwischen zwei unterschiedlichen Apps wird in diesem Framework zur Verfügung gestellt.

Das **System**: In diesem Bereich sind alle wichtigen, zum Betriebssystem notwendigen Informationen hinterlegt. Dazu gehört neben dem Datenmanagement, dem Speichermanagement und den Input/Output-Management, auch die Netzwerkverwaltung. Es lässt sich also sagen, dass dieser Bereich die Basis bildet für einen reibungslosen Betriebs des iOS.

B. Core Services

Die Abstraktionsschicht des Core Services ist eine Erweiterung des Core OS und enthält weitere, grundlegende Funktionen. Diese bieten grundlegende Klassen und Datentypen, die von den oberen Schichten verwendet werden können. Auch die Möglichkeit der Lokalisierung und der Datumsverarbeitung stehen hier zur Verfügung. Des Weiteren werden an dieser Stelle mögliche Caches-Funktionen definiert.

Seit der Version iOS 3 wird hier auch der sogenannte „in-app-purchase“ unterstützt. Dabei handelt es sich um die Möglichkeit, innerhalb einer App auf dem iPhone eine direkte Verknüpfung mit dem App-Store herzustellen und somit innerhalb der App Einkäufe zu tätigen. Dies ist besonders für Programmierer von Apps hilfreich, da einzelne Funktionen in vorhandene Apps über den App Store eingebunden werden können. Auch das Social Framework spielt eine große Rolle, da es die Verknüpfung mit den sozialen Netzwerken Twitter und Facebook herstellen kann. Da viele Apps diese Verknüpfung nutzen, kann man dieses Framework auch als sehr wichtig ansehen.

Des Weiteren werden mit Hilfe von eigenen Frameworks Dienste, wie beispielsweise der Location Service, zur Unterstützung des GPS oder auch der Media Service, zur Nutzung von Filmen und Musik zur Verfügung gestellt.

C. Media Services

Die gerade schon beschriebenen Media Services werden an dieser Stelle präzisiert. So werden mehr Typen von Audio- und Videodateien unterstützt. Außerdem werden durch diverse Frameworks in den Bereichen Audio-, Video-, Text- und Bildverarbeitung viele neue Features zur Verfügung gestellt. Auch die Animationen der unterschiedlichen Menüpunkte oder Menü Bars werden an dieser Stelle zur Verfügung gestellt.

Es handelt sich bei dem Media Service also um eine reine GUI-Schicht, die sich durch unterschiedliche Frameworks repräsentiert.

D. Cocoa Touch

Die Cocoa Touch Schicht, ist die höchste Schicht und hat somit auch den höchsten Abstraktionsgrad. Sie wird von App Programmierern primär genutzt und stellt eine Reihe von high-level Frameworks zur Verfügung. An dieser Stelle wird häufig auch in „View“ und „Controller“ unterschieden. Dies zeigt die klare Trennung der Programmierung nach dem MVC (Model-View-Controller) Prinzip, welches unter V. genauer beschrieben wird.

Als eines der wichtigsten Frameworks bei der App Erstellung wird das UIKit-Framwork angesehen. Dieses Framework bietet neben der Bereitstellung von User-Interfaces und Event-Management (bspw. Touch-Event), die Möglichkeit Nachrichten an das iOS interne Nachrichtencenter zu senden. Auch die Nutzung der Hardware (siehe oben) ist durch mehrere Frameworks gegeben.

Das Message UI Framework, ein weiteres wichtiges Framework, stellt die Funktionen im Bereich der Telefonie, SMS-Dienste und E-Mail-Versand und -Empfang zur Verfügung.

Zusammenfassend lässt sich festhalten, dass die oberste Schicht diejenige ist, die von App Programmierern am häufigsten genutzt wird. Sie stellt im Grunde alles zur Verfügung, was eine moderne App ausmacht, z.B. die Nutzung der Touch-Events. Dabei ist die Nutzung auf einer höheren Abstraktionsebene wesentlich leichter zu bewerkstelligen, als es auf den unteren Ebenen der Fall ist. Diese Bilden lediglich die Basis und stellen Schicht für Schicht immer weitere grundlegende Funktionen zur Verfügung, die, wie bereits angedeutet nicht direkt von dem Programmierer genutzt werden[6, 7].

V. GRUNDARCHITEKTUR

Die Grundarchitektur des iOS beruht auf einem weit verbreiteten Muster zur Erstellung von Software, dem Model-View-Controller, kurz MVC. Dabei werden die drei Bereiche „Model“, „View“ und „Controller“ strikt voneinander getrennt und arbeiten somit auch unabhängig voneinander. Dies trennt also die Anwendungslogik („Model“), von den dazugehörenden Darstellungen(„View“) und den Benutzerinteraktionen(„Controller“). So kann das System später leichter erweitert werden und eine schnellere Wartung ist ebenfalls ein Vorteil. Außerdem hilft die Trennung bei der individuellen und unabhängigen Nutzung der unterschiedlichen Ressourcen.

Eine Besonderheit in diesem Bereich ist die Verwaltung von Nachrichten. Diese werden mit Hilfe von einem Observer Musters repräsentiert. Hierbei werden Veränderungen eines Objektes (in diesem Fall, den Zustand einer App oder deren Nachricht) von einem zentralen Beobachter (in diesem Fall dem iOS Service) überwacht und bearbeitet. Dieses System wird zur Realisierung des Nachrichtencenters im iOS genutzt.

VI. SPREICHERVERWALTUNG

Die Speicherverwaltung des iOS, bezogen auf den Hauptspeicher, ist eine weitere grundlegende Funktion, die nicht nur die Apps benötigen, sondern auch das Betriebssystems selbst. Je nach Endgerät (z.B. beim iPhone 4 512 MB) ist der Arbeitsspeicher begrenzt auf eine gewisse Größe. Dabei wird von dem iOS selbst ein Teil dieses Speichers benötigt, um die Grundfunktionalität zur Verfügung zu stellen. Das Betriebssystem unterstützt keine dynamische Anpassung des Arbeitsspeichers, also gibt es nicht die Möglichkeit eine Auslagerungsdatei zu erstellen. Dadurch stürzen Applikationen ab und beenden sich, ohne die vorher getätigten Eingaben oder Daten zu sichern, wenn Apps mehr Speicher beanspruchen, als derzeit noch auf dem Arbeitsspeicher verfügbar. Dies ist ein großer Kritikpunkt den man an dem iOS von Apple feststellen kann.

Des Weiteren gibt es auch keine Garbage Collection, also das Entfernen von nicht mehr benötigten Inhalten durch eine automatisierte Methodik. In dem Fall, dass Objekte nicht mehr benötigt werden, verbleiben diese auf dem Arbeitsspeicher und blockieren somit wichtigen Speicherplatz. Das Management dieser unnötigen Objekte übernimmt der Programmierer bzw. Entwickler selbst, in dem er die Laufzeit dieser selbst bestimmen muss und angeben muss an welchem Punkt diese nicht mehr benötigt werden. Dies passiert durch das sogenannte reference counting, bei dem jedem Objekt ein spezielles Attribut zugewiesen wird, was einen Integer-Wert enthält. Dieser wird immer erhöht, wenn eine neue Referenz auf dieses Objekt zeigt und erniedrigt wenn eine Referenz entfernt wird. Besitzt das Objekt keine Referenzen (Integer-Wert = 0), dann wird dieses Objekt aus dem Speicher entfernt und somit wieder brauchbarer Platz freigegeben.

Außerdem besitzt jede Klasse die dealloc-Funktion, die aufgerufen wird, um ein Objekt direkt von dem Speicher zu löschen. Dies ist notwendig, wenn der Referenzzähler noch nicht auf null steht, das Objekt jedoch trotzdem vom Speicher gelöscht werden soll.

Des Weiteren wird eine Warnmeldung intern verschickt, wenn der Speicher knapp wird. Diese Warnmeldung wird in Form von einem Methodenaufruf *didRecieveMemoryWarning* ausgeführt. Mit Hilfe dieser Methode wird versucht, unbrauchbare/unnötge Daten zu löschen und somit nötigen Speicherplatz wieder freizugeben.

Zusammenfassend lässt sich feststellen, dass Apple im iOS-Betriebssystem verschiedene Modelle zur Freigabe und Nutzung des Arbeitsspeichers zur Verfügung stellt. Allein die Abwesenheit eines Garbage Collectors, der durch unterschiedliche Wege nachempfunden werden soll, ist jedoch schon fraglich. So können Apps, wenn sie mehr Speicher benötigen, als derzeit zur Verfügung steht, nicht mehr ausgeführt werden. Dies kann im schlimmsten Fall dazu führen, dass sehr viele, kurz zuvor eingegebene Daten verloren gehen. Aus Entwicklersicht jedoch, ist zu beobachten, dass es einige einfache Methodenaufrufe braucht, temporäre Daten oder nicht mehr nötige Objekte zu löschen, um wieder Speicher freizugeben[6].

VII. DATEIVERWALTUNG

Die Dateiverwaltung bildet ein weiteres, wichtiges Merkmal eines Betriebssystems. Das iOS stellt ein einfaches, windows-ähnliches System zur Verwaltung von Daten zur Verfügung. Dabei sind die iOS CoreData nicht zugänglich, stellen aber ein Dateisystem für die Apps von Drittherstellern zur Verfügung. Jede Applikation, die über das Apple eigene Applikationsarchiv (iTunes) zur Verfügung gestellt wird, erhält ein eigenes Verzeichnis in der Datenstruktur des Flashspeichers. Dabei ist die Unterverzeichnistiefe nicht begrenzt. Außerdem wird jeder Applikation auch ein temporäres Verzeichnis zur Speicherung von Daten, die zur Laufzeit benötigt werden, zur Verfügung gestellt.

Über den hauseigenen Applikationsdienst iTunes, kann der Nutzer diese Ordnerstrukturen anschauen und mit Hilfe einer Backup-Funktion auf dem eigenen PC oder Notebook sichern.

Für die iOS-eigenen Mediendienste, wie beispielsweise dem Musikplayer, werden zusätzlich Ordner angelegt, um die Musik, Filme und Dokumente zu speichern und somit auch mit Hilfe von iTunes wieder zu sichern. Vergleichbar ist diese Struktur mit den „Eigenen Dateien“ unter Windows.

Eine weitere Möglichkeit, die gerade den Entwickler anspricht, ist das sogenannte Archiving. Dabei können Objekte, die zur Laufzeit generiert werden, in einer Datei abgelegt und gespeichert werden. Diese Funktion ermöglicht dem Entwickler, gewisse Informationen schnell wiederabrufbar zu machen, ohne lange Umwege über das Extrahieren und Zusammensetzen der Objektinformationen zu gehen.

Unabhängig von dem Dateisystem, existiert des Weiteren ein Datenbanksystem, um relevanten Informationen zu speichert. Auch hier haben die Apps von Drittanwender Zugriff. Eine App kann also eine oder mehrere Datenbanken anlegen, um Informationen dort zu speichern. Der Zugriff auf diese Datenbanken erfolgt mit dem, im iOS enthaltenen Service SQLite. Dies ist ein häufig verwendetes Datenbanksystem für mobile Systeme, welches einen Zugriff auf die Datenbanken direkt aus dem Quellcode einer App ermöglicht (SQLite-Libary). Hierzu werden die, aus SQL bekannten Abfragen genutzt [6, 7, 8].

VIII. APPS FÜR DAS IOS

In den vorherigen Kapiteln wurden grundlegende Informationen über das Handybetriebssystem iOS gegeben. Immer wieder war die Rede von den sogenannten Apps. Im Folgenden soll nun eine Standard-App genauer betrachtet und eine Grundstruktur erkannt werden.

Zu Beginn steht eine gewisse Anzahl an Templates zur Verfügung. Viele Apps werden so nach dem gleichen Muster erstellt und bieten somit auch den klassischen „iPhone-Look“. Das wohl meist gewählte Template ist hierbei das Window-based-Template. Die GUI einer App wird somit von dem Fenster an sich repräsentiert und nehmen die Möglichkeit der direkten Eingabe an. Auch das annehmen der indirekten Eingabe (bspw. mit Hilfe des Gyroscope) können angenommen werden, wenn der Entwickler dies benötigt.

Des Weiteren ist die klassische GUI einer iPhone-App aus unterschiedlichen Views zusammengebaut. Diese unterschiedlichen Elemente, können beispielsweise eine Navigationsleiste, Buttons oder auch reine Textfelder sein. Dem Entwickler ist freigestellt wie viele unterschiedliche Elemente bzw. Views er auf dem Bildschirm anzeigen lässt. Jedoch ist aus der Übersichtsanforderung eine indirekte Grenze gesetzt (siehe Abbildung 1).

Abbildung 1: Klassischer Aufbau einer App im iOS.

Damit die nun erstellte App nicht nur aussieht wie eine klassische iPhone App, sondern dem Nutzer auch die Auswahl der gewohnten Eingabemöglichkeiten zur Verfügung steht, müssen die Apps diese Eingaben abfangen. Dies passiert so, dass die unterschiedlichen View-Elemente, die Nutzereingaben direkt verarbeiten und weiterleiten können an die folgenden Schichten (MVC-Schichtenprinzip wie oben bereits beschrieben). Dies gilt jedoch nur für die sogenannten Touch-Eingaben, die der Nutzer über den Touchscreen am Endgerät durchführt. Die verschiedenen Sensoren, wie beispielsweise das Schütteln des Gerätes oder die Positionsermittlung geschieht nicht direkt auf der View-Ebene, sondern wird durch die Controller abgefangen und verarbeitet. Somit können die Apps nun auch Eingaben annehmen.

Die restliche Funktionalität entsteht dann im Model und Controller (nach MVC). Grob gesagt, enthält die Model-

Eben dann die eigentliche Logik der App und die Controller steuern diese dann an und benutzen sie.

Zur Erstellung und Entwicklung einer App im iOS gibt es viele unterschiedliche Softwarelösungen, die aber größtenteils nur für das Apple Betriebssystem MacOS zur Verfügung stehen. Auch Apple selber stellt ein Developer Programm zur Verfügung. Des Weiteren ist das emulieren eines iOS-System durchaus möglich und auch sinnvoll um die erstellte App zu testen und zu debuggen.

Am Schluss jeder Entwicklung steht das Release der programmierten App. Im speziellen Falle der erstellten iOS-App heißt das, dass die App mit Hilfe des Apple AppStore vertrieben wird. Eine andere Möglichkeit der Verbreitung der eigenen App besteht nicht.

Nach dem Hochladen der App in den AppStore dauert es ca. eine Woche, bis die App dann zum Download zur Verfügung steht. Apple sichert sich 30% des Erlöses der App, falls diese Kostenpflichtig ist. Jedoch werden Dinge, wie die Vertriebswege und Marketing kostenlos zur Verfügung gestellt, z.B. der Speicherplatz und die Angebotsseite für die App.

Abschließend lässt sich festhalten, dass eine App für iOS in großem Maße standardisiert ist und gewissen Grundsätzen entspricht. Natürlich gibt es gerade im Bereich der grafisch anspruchsvolleren Apps, wie beispielsweise 3D-Spielen andere Templates bzw. komplett andere Vorgehensweisen. Jedoch ist die Standard-App häufig sehr ähnlich strukturiert.

Eine große Kritik lässt sich an der hohen Beteiligung von Apple an den Apps, die im AppStore verkauft werden, anbringen. So gehen knapp ein Drittel des Erlöses an Apple, die dafür lediglich die Plattform schaffen und keine andere Möglichkeit der Distribution zulassen. Die Entwickler sind an dieser Stelle sehr stark gebunden und müssen sich somit mit Apple als Distributor abfinden. [7]

IX. Ausblick und Kritik

In diversen Internetforen im Bereich der iOS-Entwicklung wird gemutmaßt, dass sich eine revolutionäre neue Version des iOS bereits in der Entwicklung befindet. Der Nachfolger mit dem Namen iOS 7 soll demnach im Laufe des Jahres 2013 vorgestellt werden. Wie immer bei Apple, ist der Termin streng geheim gehalten, um die Präsentation somit medienwirksamer inszenieren zu können. Aber eben gerade auf Grund solcher Medienstrategien und ausgeklügelten Werbestrategien hat es Apple mit iPhone und iPad erst zu richtig großem Erfolg im Bereich der Smartphones gebracht. So ist es dem einzigartigen und revolutionären Design des iPhones und dem, optimal auf das Gerät zugeschnittenen Betriebssystems iOS zu verdanken, dass man eine Vormachtstellung in diesem Sektor hat.

Jedoch muss man auch Kritik üben an dem Betriebssystem bzw. dem generellen Vorgehen im Bereich der Software-Entwicklung. So gibt Apple keinerlei Informationen über den Source-Code ihrer Betriebssysteme preis. Auch das Fehlen einer Garbage Collection spricht nicht für ein entwicklerfreundliches System. Dennoch gibt es für kein anderes Handybetriebssystem mehr Apps als für iOS. Dies ist nur damit zu erklären, dass Apple in diesem Bereich der Marktführer ist und sich deshalb sehr schnell und gut Geld verdienen lässt, mit der App-Programmierung. Denn mittlerweile ist es schon ein Marktnachteil für ein großes Unternehmen, wenn es keine iPhone-App besitzt, die über diese Firma oder deren Produkte informiert, bzw. einen Shop oder Service anbietet.

Ein weiterer Kritikpunkt direkt an dem Unternehmen Apple sind die hohen Preise der Endgeräte. Diese liegen in einem sehr hohen Segment, obwohl die Leistung häufig nicht sehr viel höher ist als es bei Vergleichsgeräten der Fall ist. Dies ist allein damit zu erklären, dass es mittlerweile ein Statussymbol ist, ein iPhone zu besitzen und dies auch durch Apple so propagiert wird.

Schlussendlich kann man sagen, dass es sich bei dem Handybetriebssystem iOS zwar um ein sehr gut funktionierendes und schnelles System handelt, welches dem Nutzer viele Möglichkeiten bietet und welches optimal auf die Hardware des Endgerätes abgestimmt, jedoch inkompatibel zu Geräten, die nicht aus dem Hause Apple stammen, ist. Des Weiteren ist der geschlossene SourceCode und die teure Anschaffung und die damit verbundene Kompatibilität mit anderen Betriebssystemen, häufig ein Stein im Wege eines App-Programmierers für iOS.

X. References

Diese Arbeit wurde mit Hilfe unterschiedlicher Quellen erstellt. Da sehr schwierig war, Literatur in diesem Bereich zu finden, sind alle Quellen aus dem Internet entnommen. Insbesondere standen an dieser Stelle die allgemeinen Developer Seiten des Herstellers Apple zur Verfügung.

[1] Marktanteile: http://www.squaredenker.com/news/iphone-marktanteile-ios-unter-smartphone-betriebssystemen-weltweit-die-nummer-eins.html#!prettyPhoto, Zugriff 27.01.2013

[2] Nutzung von Smartphones und mobiles Internet: http://www.aquarius.biz/de/2011/10/10/acta-2011mobile-internetnutzung-sprunghaft-gestiegen/, Zugriff 27.01.2013

[3] Opel iPhone-App wird vorgestellt: http://www.welt.de/motor/news/article106368554/Schnell-verbunden.html, Zugriff 27.01.2013

[4] Allgemeine Informationen zum iOS: http://www.apple.com/de/ios/, Zugriff 03.01.2013

[5] Apple Produktinformationen iPhone 5: http://www.apple.com/de/iphone/specs.html , Zugriff 03.01.2013

[6] Apple Developer Center: https://developer.apple.com/devcenter/ios/, Zugriff 03.01.2013

[7] Dateiverwaltung im iOS: http://www.iosdevgermany.de/tutorial/sqlite-tutorial-deutsch-datenbanken-auf-dem-iphone/, Zugriff 05.01.2013

Smartphone Operating Systems: Maemo

Sebastian Reichel
University of Oldenburg
Oldenburg, Germany
sre@ring0.de

***Abstract*—This paper is about the Maemo 5 operating system and the only smartphone ever released with it: The Nokia N900.**

The first part describes how different hardware components of the N900 are connected to the system and which software parts are involved to provide user access. It focuses on low-level interfaces.

In the second section parts of Maemo's system software is described, which is independent of specific hardware components. The focus lays on high-level interfaces.

The last part focuses on ideas and technologies of Maemo, which have already been taken over by other projects or may be taken over in the future.

I. Introduction

Maemo is a mobile operating system (OS) originally designed for Nokia's internet tablet series. Initially, the operating system had no phone related features at all. Instead, it was merely an OS used for web browsing via wireless LAN, making notes, etc. The OS itself is based on many popular open source software components.

In 2009, Nokia built an internet tablet called N900, which contains hardware for using mobile phone networks and therefore updated the operating system by adding support for this kind of hardware.

Nokia abandoned their platform two years later in favour of MeeGo. Since then a group of developers from the community continued supporting Maemo by providing updates.

This paper will focus on Maemo 5 (*Fremantle*) and the Nokia N900, since Fremantle is the only Maemo release running on a smartphone and the Nokia N900 is the only device using it.

In the Section II the underlying software architecture of Maemo is described. Section III focuses on the Nokia N900's hardware components including information how they are connected to the system. This section also describes the software parts included in the Maemo OS, which use the hardware. Important software delivered as part of Maemo, but not directly controlling hardware components will be described in Section IV. Section V focuses on the future of Maemo. It contains an outlook on Maemo's successors and forks.

II. Architecture

Fremantle is based on the Linux kernel 2.6.28[22]. The main changes to the standard kernel are additional hardware drivers. The core system is based on Debian[8]. In particular, Maemo is using Debian's packaging system dpkg[9] and apt-get[2].

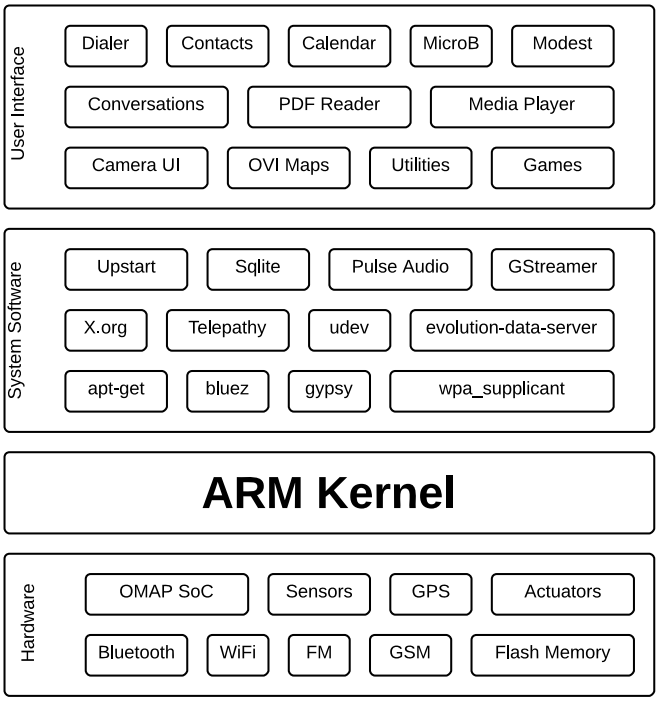


Fig. 1. Architecture

The complete user interface (UI) is running on a standard Xorg server[39] using matchbox[25] as a window manager. All system software is using the GIMP-Toolkit (GTK+)[13] from the GNOME project[11]. In addition to the default GTK+ widgets, Nokia wrote their own set of widgets, which are fitted for usage on mobile devices. This set of widgets is called Hildon[16].

Other popular open source software used in Maemo is the Telepathy framework[34], which is the Maemo basis for all Real-Time communication.

It is using GStreamer[12] for demultiplexing and decoding files, and PulseAudio[32] for routing and mixing audio streams.

Many of the components use the D-Bus[7] for inter-process communication, which is a powerful network based protocol interchanging XML messages.

Figure 1 illustrates the key elements used by the Maemo operating system.

III. Hardware Interface

This section describes the most important hardware components of the Nokia N900, how they are connected to the CPU, and how they are exposed to the system by the Linux kernel.

A. Flash Memory

The Nokia N900 contains two different kinds of memory for data storage. It has a 256MB NAND memory chip and a 32GB embedded MultiMediaCard (eMMC) chip. The NAND memory is much faster than the eMMC memory, but it has the disadvantage of providing too little space for the operating system and the user's data.

Nokia implemented a new driver for the Linux kernel, which is used by default on the Nokia N900 for the NAND memory. It can only be used on raw flash memory devices resulting in the driver's name, Unsorted Block Image File System (UBIFS).

Maemo's root filesystem is stored on the NAND memory, with 2GB of the eMMC mounted to /home and /opt being linked to /home/opt. Most software packages have their data stored in /opt, since the remaining space on the NAND is very limited. The remaining eMMC memory is FAT32 formated and mounted to a directory named "MyDocs", that is located in the user's home directory.

This directory contains the user's private data like music and photos from the camera. When the phone is connected via USB, Maemo provides the option to unmount this directory from the system and export it to the Personal Computer (PC) as a mass storage device.

B. Modem

The modem is produced by Nokia under the codename Rapuyama. The modem has its own 128MB DRAM / 128MB NOR combo chip running its own firmware [30]. It is connected to the OMAP3430 over Synchronous Serial Interface (SSI) [30]. In contrast to other smartphone designs, the audio lines are not directly connected to the sound chip, but they are sent over the SSI connection [30]. Also unlike other smartphones, Nokia's modem does not use standard AT commands to communicate with the main processor. Instead, Nokia defined their own binary protocol, called ISI [22].

The linux kernel provides drivers for the SSI connected device. Userspace can access the device via /dev/hsi (it stands for High Speed Interconnect, which is the vendor independent successor of OMAP's SSI). This file is used to send and receive audio streams. For other communication the kernel provides another abstraction layer, which is called Phonet. It offers access to the modem via virtual network devices [22]. There is an wireshark protocol analyser [17] available, which had been written by the community, and is based on reverse engineered information.

C. Audio & Video

Audio and video are handled by GStreamer [12]. Nokia provided GStreamer plugins, which offload the decoding of different video and audio codecs on the OMAP's C64x+ Digital Signal Processor (DSP). This makes the N900 capable of playing H264 encoded videos with a resolution up to 1280x720 pixels.

The decoded audio is then passed to PulseAudio [32], which manages many things:

- Mixing multiple audio streams together
- Setting up different volume controls
- Providing different audio profiles (speaker, earphone, headset, ...)
- Handling audio related events (plugin of speaker etc)
- Routing audio between modem and sound chip
- Acoustic Echo Cancelling

PulseAudio must be able to handle packages with hard timing constraints: GSM and 3G audio frame size is 20ms. The Linux kernel provides access to the audio hardware to the userland on the usual ways via the Advanced Linux Sound Architecture (ALSA) [33]. On the hardware side, Texas Instrument TWL4030 is used. It is connected over multiple interfaces to the OMAP processor.

All Video related work goes over the Linux's Framebuffer interface. Newer (released after Maemo) kernels also provide a Direct Rendering Manager (DRM) interface. The framebuffer driver then passes the data to another framework, which is called OMAP Display Subsystem (OMAPDSS), which forwards the images to either the built in display panel, or an external display.

D. Camera

The N900 contains two different cameras. One with lower resolution of 1 Megapixel (MP) faces the user and is intended for video calls. Another is intended as photo camera and has a 5MP resolution.

Since the OMAP 3430 provides only a single Camera Serial Interface (CSI), both cameras are connected to a switch, which in turn is connected to the CSI. Thus only one camera can be used at point in time.

In the Linux kernel, the Cameras are exposed via the Video4Linux 2 (V4L2) interface. This interface is also used for TV cards, webcams, etc. on desktops. Nokia wrote a GStreamer plugin for the cameras, so that it can be easily used by userland software.

E. WLAN

The WLAN chip used in the Nokia N900 is connected via Serial Periphal Interface (SPI) [30]. The chip supports WEP and WPA crypto algorithms resulting in less CPU load. The kernel provides device access via a standard network interface.

Maemo is using wpa_supplicant in the userspace (the same daemon is used by the major Linux distributions). On the other hand the Graphical User Interface (GUI) is a custom piece of software.

F. Bluetooth

The used Bluetooth chip is a Broadcom BCM2048. It is connected via one of the OMAP's serial lines. As a special feature it also contains an FM radio receiver. The chip's audio

data is not routed over the CPU. Instead, the audio lines are directly connected to the TWL4030. The TWL4030 can be configured to route audio to or from the Bluetooth chip.

The Bluetooth part of the chip is handled by the respective subsystem in the Linux kernel and the Bluez daemon in the userland. This is again exactly the same as in the major Linux distributions.

The FM receiver part is handled by the V4L2 interface. Maemo does not include any software using this feature, but there are third party applications available in the repositories.

G. GPS

The Nokia developers connected the GPS to the modem using Inter-Integrated Circuit (I2C) and a 4 bit data connection instead of connecting it directly to the main processor. The used chip is a GPS5030 from Texas Instruments marketed under the name Navilink 5 [30].

The kernel has no special support for the GPS. It is just a normal ISI message sent to the Phonet network interface. On the userland side, a small custom library, called liblocation [21] provides access to GPS related information.

H. Keyboard & Touchscreen

The N900's keyboard is a simple button matrix connected to General Purpose Input Output (GPIO) pins of the OMAP processor. There's no special chip involved. The button mapping is done in the Linux kernel. In contrast to the keyboard, the touchscreen has its own chip called TSC2005. It takes care of handling the touchscreen's matrix and is connected over SPI. In addition to the SPI interface the touchscreen uses an GPIO pin for sending interrupts to the controller when there is a touch event [22].

The kernel exposes each of the devices as Human Interface Device (HID). The userspace software receives keyboard and cursor events from the kernel, which is exactly the same as on normal computers. The X-Server has custom support for the Keyboard to support the special keyboard layout using custom key combinations [22], [1].

I. Power Management

Power management is an important topic in the mobile phone sector. On the one hand, users want a phone, which has very short response times, but on the other hand, the phone's battery should last as long as possible. For this, similar to PC systems, the phone's CPU frequency is dynamically clocked. Linux uses the same framework (cpufreq) on PCs and mobile phones. But on smartphones, one is also able to do dynamic voltage and frequency scaling (DVFS) on slave devices. A new framework has been added to the Linux kernel (devfreq) to support this [14]. This framework would support the N900 hardware, but it is not part of Maemo.

Maemo is using the OMAP's SmartReflex system on the N900, though. This technique, which is called adaptive voltage scaling (AVS) by the kernel developers adapts the voltage of different chip parts depending on static factors (chip manufacturing process) and dynamic factors (temperature depending performance) [20].

Another technique, which is part of the power management is called Dynamic Power Switching (DPS) by Texas Instruments, which is supposed to put hardware, which is currently not in use into a low power state. For example, a processor can be put into a low power state while it is waiting for a DMA transfer to complete [29].

The last power management realated key element is called static leakage management (SLM). In contrast to DPS it does not put a section of the system on a chip (SoC) in a low power state, but initiates a standby or device-off mode of the whole system. All internal states are stored into internal memory (or external memory for device-off) and restored on power up. This results in a much faster boot compared to a cold boot [29].

J. Battery Management

The N900, like many smartphones, does not have an independent autonomous battery/charging management chip. Instead, most controlling is done in software components of the OS. This means, that the phone must be booted for charging correctly!

For battery management two chips are used. The bq27200 is an I2C connected chip, which is used for monitoring the smartphone's Lithium Ion (Li-ion) battery [22], [4].

There's another I2C connected chip, called bq24150a. It is used for correctly charging the telephone's battery [3]. Since charging is done via the N900's Universal Serial Bus (USB) port, more chips are involved in charging. In particular the isp1707a is important, which detects chargers on the USB port. It is needed to verify, that the host system's USB port is not damaged by draining too much power from it.

In Maemo, battery management is done by a closed source userspace daemon, which has been named Battery management entity (BME) by Nokia. According to some Nokia developers, BME is kept closed source, because it is using high efficient charging algorithms, which are patented. Apart from that Nokia fears, that changing the code may result in exploding batteries [5].

Since then multiple hackers have been working on open-source drivers for the Linux kernel. The recent mainline kernel contains drivers for the single chips, but does not yet contain a master driver controlling them.

When the phone is plugged into a USB port or connected to a USB charger, it automatically turns on and boots into a minimal system, which runs the BME daemon described above. This is needed, because the chips cannot charge the battery safely without being monitored by software.

K. Universal Serial Bus

As mentioned in the battery management section, multiple chips are involved in handling the USB port. The main USB part is done by the OMAP itself. It contains an Intelectual Property (IP) core from Mentorgrafix called MUSBMHDRC-core, which is supported by the Linux kernel. The physical layer is connected to the isp1707a.

The TWL4030 is also connected to the USB port, to wakeup the system if USB power is attached to the port. The reasons for this are described in the previous battery management section.

The last chip involved is the bq24150a battery charging chip. It can not only be used as battery charger, but also in the reverse way to give power to the USB plug [3].

By default, Maemo only supports using the N900 as USB slave device, which means it can only be connected to a computer, but not to USB periphery. When the phone is connected, Maemo offers the user to export the smartphone's memory as a flash drive or to connect the phone as USB network device.

Some hackers from the community added USB host support later, which is available as a third party software for Maemo. Using their kernel, one can directly connect to and use USB devices like pen drives or keyboards with the telephone.

L. Other Hardware

Also connected is a distance sensor giving boolean outputs via a GPIO pin. Maemo's kernel exports it as a virtual file. Unlike the IR diode, the distance sensor is used by default software: The phone application locks the screen, one the proximity sensor senses anything, so that no UI elements are accidently activated by the user's face during calls.

The N900 contains another sensor used by the default software. It is a brightness sensor, called TSL2563, which is connected via I2C. The default Maemo software stack uses it to auto-adjust the display brightness relative to the user's settings.

While these things are also found on many other phones, Nokia built some hardware components into the N900, which are unusul for smartphones.

The telephone contains an infrared (IR) diode, which is simply connected to an GPIO of the processor. Maemo's kernel contains a driver for this LED, but there is no default installed software, that makes use of it. In the repositories, there is a third party software available, which uses the LED for controlling TVs, though [22].

In addition to the FM receiver contained in the Bluetooth chip, Nokia also built in an FM transmitter called Si4713. It is connected via I2C and exported via the V4L2 API by the Maemo kernel. The default media player shipped by Maemo supports this piece of hardware, so that music can be streamed to a nearby FM receiver (for example a car radio) [22].

The 3.5mm jack socket can be used not only for headphones and headsets, but also for streaming PAL or NTSC videos. The default software automatically mirrors the phone's display to the external connected TV. This feature can be used out of the box with the media player for watching videos on TV using the N900 as media player.

IV. System Software

This section describes important software, which is shipped with the operating system, but independent of specific hardware components.

A. Booting Software

Maemo is using Ubuntu's upstart service for booting the system [37]. In contrast to the default SysVinit of unix systems, it has the advantage of including dependency information between system services. Thus it can start different system services parallel instead of starting them sequentially like the default SysVinit. The result is a fast booting system.

During the early boot phase, some script from Nokia read out flags given to the kernel from the bootloader, which in turn reads them from the OMAP chip. This flags contain the information why the system has been booted (e.g. because of an USB plug event).

When the system is connected to a USB power source while it is turned off, it automatically wakes up and boots. The script described in the previous section does not boot the system into the normal state. Instead it boots a minimal system, which only charges the battery. When the users tries to start the system using the power button, it reboots into the normal mode.

B. Browser

Maemo 5 is shipped with a browser called MicroB [26], which has been developed specifically for this operating system. It is based on the previously described Hildon widgets and uses Mozilla's Gecko engine. In contrast to other mobile operating systems, Adobe's Flash plugin is not only available, but is bundled together with the browser.

C. Mailer

Nokia also wrote their own E-Mail client, which is called Modest and is based on the tinymail framework [27]. It is a lightweight mail client supporting only basic features. Push mail is supported, but only when using Nokia's mail server.

D. Media Player

Maemo's media player supports playing music and movies from local storage (ca. 30 GB on the N900) or micro Secure Digital (μSD) card. It also supports playing audio streams from the web (Internet Radio). There is no support for the FM receiver integrated in the bluetooth chip, but it can be used for playing media provided by Universal Plug and Play Audio Video (UPnP AV).

The media player has integrated support for enabling the N900's FM transmitter to send audio as FM stream to a radio next to it. The player also supports output over the N900's TV out cable, so that videos can be watched on all television sets having standard composite connectors.

E. Maps

Maemo 5 is shipped together with Ovi Maps. Starting the application automatically enables the GPS chip and downloads the needed map data from Nokia's servers.

It has integrated support for pedestrian and car routing, but it does no voice announcements.

F. Real-Time Communication

One of the features first found in Maemo is the handling of Real-Time communication. Maemo 5 is shipped with Telepathy as system service for handling all Real-Time communication. Telepathy can be extended by plugins to support more protocols.

The N900 is delivered with different plugins installed by default, which support voice and text messaging as well as video chats. Some of the plugins also support conference calls using the following protocols:

- Skype
- Session Initiation Protocol (SIP)
- Extensible Messaging and Presence Protocol (XMPP)

Support for normal phone calls and Short Message Service (SMS) is also provided by telepathy plugins.

Other plugins available from the Maemo repository include support for Microsoft's MSN service and AOL's ICQ service as well as other instant messaging protocols used around the world.

This results in a single user interface used independent of the protocol used. The dialer supports changing the call type.

V. Future

As mentioned in the introduction, Maemo's development has been discontinued. But this does not mean that its software components are no longer being developed. As described in the previous sections, most software components are common open source projects (e.g. Xorg).

The concept of providing access to GSM features (SMS, phone) via Telepathy has been reimplemented as an open source project by Nokia and Intel. This project has been called oFono and telepathy-ring, and is used for example by MeeGo.

Most parts of the User Interface are no longer actively being worked on, since Meego used Qt based Software in favour of the GTK+ based Hildon Desktop Environment. Thus, development on Hildon Desktop, Application Manager and Control Panel is mostly stalled.

It needs to be mentioned though, that there is a small group of developers, who still write software updates for Maemo as used to be on the Nokia N900. They call their work Maemo Community Seamless Software Updates (CSSU). The CSSU repository contains Hildon updates for supporting portrait mode of the desktop and many applications.

The CSSU repository also contains rewrites of closed source Maemo software, which adds features missing in the closed source counterpart. The most famous example is the open media player, which has added portrait support and a car mode.

Another rewritten piece of closed software is the camera application. The opensource version has added support for keyboard bindings, manual white balance setup, timer based capturing, and focus distance information giving [6].

As mentioned in the introduction Maemo development has been stopped by Nokia in favour of MeeGo. MeeGo is a combination of Nokia's Maemo and Intel's Moblin platform. The main differences between Maemo and Meego are:

- Usage of QT instead of the GTK+ based Hildon framework. Nokia had bought Trolltech (the company behind QT) previously.
- Usage of a RPM instead of APT based package manager.
- Completly redesigned user interface.
- Operating Systems variants for non smartphone hardware like tablets or in-vehicle systems.

MeeGo's collapsed only one year after its first release, after Nokia has changed their strategy in February 2011. At the beginning of 2012, a new phone operating system was created, which is based on MeeGo: Tizen.

In contrast to MeeGo, Tizen makes heavy use of the WebKit engine, and most applications are simply based on HTML5. Native applications are supposed to use the Enlightenment Foundation Libraries (EFL), but Qt and GTK+ applications are also supported. Tizen is supported by the Linux Foundation, Samsung, and Intel [35].

Another operating system related to Maemo is called Mer. It has been started with the aim to provide a completly free alternative of Maemo for the Nokia N800 and N810 internet tablets. When Maemo 5 was released for the N900, they added another goal: Porting as much of it to the N8x0 devices as possible.

When MeeGo was released, Mer's new goal was to build MeeGo for the N8x0 devices. The Mer project announces its plan to continue developing Mer based on MeeGo, after the MeeGo development has been stopped in favour of Tizen. This transformed the project effectivly into MeeGo 2.0 [28].

Obviously the Linux kernel is still being worked on. Many of the device drivers written by Nokia have been merged into the mainline kernel. One example for a driver written for Maemo, which got very popular and is activly being worked on, is UBIFS. The least recently released stable kernel (3.7) contains a huge changeset adding fastmap support [38].

Some problems and disturbances, which were noticed by existing code of Maemo and other Linux based smartphones are now being worked on. One of the big disadvantages of the ARM platform is that many devices are connected via buses, that do not support slave device auto detection and identification. Thus, the kernel has board specific source code for each device it is supporting. For example, the current Linux kernel contains almost 1500 lines of source code in "arch/arm/mach-omap2/" to describe the Nokia N900. This is obviously a problem if many devices should be supported, because the kernel size increases very fast.

The Linux kernel developer's solution to this problem is called Flattened Device Tree (FDT). The idea of FDT is describing the hardware connections in a configuration file, which is given to the running kernel by the bootloader starting the kernel. One of the first things the kernel does after being loaded is parsing the FDT file to get information about the connected devices.

Another problem identified in the ARM kernel is missing support for using the same kernel image on different ARM platforms. For example, one has to build different kernels for an NVIDIA Tegra based platform and the Texas Instruments

based OMAP platform. The kernel developers are currently putting effort into supporting kernel images running on multiple platforms. One of the key features needed for this is called Common Clock Framework. It tries to unify the APIs used on different ARM platforms for managing system clocks [10].

Other improvements are based on the ARM hardware platform itself. One of the recent changes is called big.LITTLE, which combines high speed ARM cores with low power cores in a single chip. The operating system is supposed to enable the high speed cores when high performance is needed and the low power ones when the system is idle. The Linux kernel developers are still working on proper support for asymmetric multiprocessing. The problem here is not the how to switch the cores, which can already be done, but the question when the switching should be done [31].

The next important update is the new ARMv8 platform. ARMv8 brings a completly new instruction set called AArch64, which is 64-bit based, while still supporting the old ARMv7 instruction set. Support for the new instruction set has already been implemented by kernel developers hired by ARM Ltd. [24].

As mentioned in Subsection III-J userspace interaction is needed for charging the Li-Ion battery of some smartphones. This means that the device cannot be suspended while it is charged. Recently, Samsung added a new charging manager framework, which is supposed to do the needed polling. It also supports charging while being suspended. For this, it sets a timer wakeup interrupt for the desired polling time. When the device wakes up, the charger manager checks if it is responsible for the wakeup, handles it, and sends the device back into suspend before it has completly woken up [15].

A feature supported by the Nokia N900 hardware, but not by the Maemo kernel, has been added to the mainline kernel since then: hardware crypto acceleration. The OMAP processors have a crypto acceleration unit called M-Shield, which accelerates the Data Encryption Standard (DES), Triple DES, Advanced Encrytion Standard (AES), RSA, and DSA. It also supports acceleration of SHA-1 and MD5. Moreover it includes a random number generator [18, 19].

VI. Application Development

Developing applications for Maemo is not very different from developing them for normal Linux computers. Many different programming languages are available for the system. Most system parts are implemented in C and some UI elements are implemented in Python. There is an SDK, which contains a cross compiler for the ARM chip used in the Nokia N900.

For better integration with other parts of the system, it is recommended to use the Hildon Widgets [16] and low-level interfaces described before. This is just a recommendation, though. It is also possible to write applications using Qt or EFL. There toolkits just don't use Maemo's UI theme out of the box.

Marc-Hendic Luehr's paper [23] presents more about this topic.

VII. Conclusion

Maemo is an operating system, which brought new ideas and concepts into the market of mobile operating systems. Even though the Maemo is considered as being dead since Nokia dropped support for it, the community keeps it alive.

Multiple forks and successors came after MeeGo, but none gained as much popularity as Android, iOS, or even Windows on mobile devices. This is not very surprising when taking into account that, for most of these systems, there exists only a single smartphone being shipped with it. From the developers perspective, it is not worth to port or write their application for an operating system running only on a single phone, and most users are not interested to buy phones or use operating systems without a big pool of applications.

Even though the average mainstream users are not interested in these operating systems, there are still some marginalized groups using them.

Acknowledgment

Thanks to all developers from Nokia, Texas Instruments and the community for keeping the system alive by mainlining the N900 kernel drivers, rewriting closed source software parts, and maintaining the different components. Also many thanks to Pali Rohár for his efforts in adding N900 support to the U-Boot[36] bootloader and maintaining the Free Fiasco Firmware Flasher (0xFFFF) for the N900.

References

[1] file on device: /usr/share/X11/xkb/symbols/nokia_vndr/rx-51.

[2] *apt-get*. URL: http://man.he.net/?topic=apt-get\&section=8.

[3] *bq24150a*. URL: http://www.ti.com/product/bq24150a.

[4] *bq27200*. URL: http://www.ti.com/product/bq27200.

[5] *Closed Maemo Software*. URL: http://wiki.maemo.org/Why_the_closed_packages#Specific_reasons_for_packages.

[6] *CSSU*. URL: http://wiki.maemo.org/Community_SSU.

[7] *DBus*. URL: http://dbus.freedesktop.org.

[8] *Debian*. URL: http://www.debian.org/.

[9] *dpkg*. URL: http://man.he.net/?topic=dpkg\&section=1.

[10] Jake Edge. "A common clock framework". Dec. 2011. URL: https://lwn.net/Articles/472998/.

[11] *Gnome*. URL: http://www.gnome.org.

[12] *GStreamer*. URL: http://gstreamer.freedesktop.org.

[13] *GTK*. URL: http://www.gtk.org.

[14] MyungJoo Ham. "Devfreq, DVFS Framework for Non-CPU Devices". Sept. 2011. URL: https://lwn.net/Articles/460973/.

[15] MyungJoo Ham. "Linux Conference Europe 2011: Charger Manager". In: Oct. 2012. URL: http://elinux.org/images/c/c6/Elce11_ham.pdf.

[16] *Hildon*. URL: http://maemo.org/api_refs/5.0/5.0-final/hildon.

[17] *ISI Wireshark-Plugin*. URL: http://sre.ring0.de/isi-wireshark-plugin.

[18] Dmitry Kasatkin. “omap-aes: OMAP2/3 AES HW accelerator driver”. Aug. 2010. URL: http://article.gmane.org/gmane.linux.kernel.cryptoapi/4656.
[19] Dmitry Kasatkin. “omap-sham: OMAP SHA1/MD5 driver”. Apr. 2010. URL: http://article.gmane.org/gmane.linux.kernel.cryptoapi/4223.
[20] J Keerthy. “PM: Create the AVS(Adaptive Voltage Scaling)”. Apr. 2012. URL: https://lkml.org/lkml/2012/4/26/258.
[21] *liblocation*. URL: http://maemo.org/api_refs/5.0/5.0-final/liblocation/.
[22] *Linux Kernel*. URL: http://kernel.org/.
[23] Marc-Hendic Luehr. *Maemo - Special Applications and Programming*.
[24] Catalin Marinas. “AArch64 Linux kernel port”. Aug. 2012. URL: https://lwn.net/Articles/511242/.
[25] *Matchbox*. URL: https://matchbox-project.org.
[26] *MicroB*. URL: http://browser.garage.maemo.org.
[27] *Modest*. URL: http://modest.garage.maemo.org.
[28] Carsten Munk. “MeeGo Reconstructed - a plan of action and direction for MeeGo”. Oct. 2011. URL: http://lists.meego.com/pipermail/meego-dev/2011-October/484215.html.
[29] Arthur Musah and Andy Dykstra. *Dynamic power management techniques for multimedia processors*. July 2008. URL: http://www.eetimes.com/design/power-management-design/4012243/Dynamic-power-management-techniques-for-multimedia-processors.
[30] *Nokia N900 RX-51 Schematics*. 2.0. Nokia. Sept. 2009.
[31] Nicolas Pitre. “Linux support for ARM big.LITTLE”. Feb. 2012. URL: https://lwn.net/Articles/481055/.
[32] *Pulse Audio*. URL: http://www.pulseaudio.org.
[33] Jyri Sarha. “Pracitcal Experiences from Using Pulseaudio in Embedded Handheld Devices”. In: Sept. 2009. URL: http://linuxplumbersconf.org/2009/slides/Jyri-Sarha-audio_miniconf_slides.pdf.
[34] *Telepathy*. URL: http://telepathy.freedesktop.org.
[35] *Tizen*. URL: https://www.tizen.org.
[36] *U-Boot*. URL: http://www.denx.de/wiki/U-Boot.
[37] *Upstart*. URL: http://upstart.ubuntu.com.
[38] Richard Weinberger. “UBI: Fastmap request for inclusion”. Sept. 2012. URL: https://lwn.net/Articles/517422/.
[39] *X.org*. URL: http://www.x.org/.

Maemo - Special Applications and Programming

Marc-Hendric Lühr
Email: marcluehr@googlemail.com
Carl-von-Ossietzky Universität Oldenburg
Seminar Handy Betriebssysteme (WS2012/13)

***Abstract*—Having a full open source system for mobile devices offers some interesting advantages. It supports full customization of the system and the freedom to write new applications for various purposes. Unlike IOS and Android, Maemo grants full access to the system and to its source code.**

This paper describes special applications for Maemo which are ready to use. Some special applications are already installed on the system, other applications must be installed by the user. Also, the writing of programs is described. The huge amount of different sensors and actors opens new perspectives for better interaction and more efficient work with the mobile device. Nevertheless, writing applications requires knowledge about system interfaces. On the one hand the programming language can be choosen by the programmer according to his personal skills but on the other hand there are recommendations and already implemented interfaces for specific programming languages. These are described as well.

I. Introduction

Maemo is an operation system for mobile devices developed by Nokia [1]. The first release was in November 2005. The current version is Meamo 5 with the codename *Fremantle*. The last official update was in October 2011. Since this time, Nokia stopped the development of Meamo and left it to the community. Nokia merged Meamo with Moblin to MeeGo whose development Nokia also has discontinued.

Maemo is built on Debian Linux using a customized kernel, the Gnome Desktop environment and free software. The source code of Maemo itself is freely available and can run on nearly every smartphone or tablet pc with matching minimum requirements. The packages are build for the ARM plattform, so the smartphone should also have an ARM processor but the x86 architecture is also supported. Nokia developed Maemo for internet tablet pcs which have similar specifications to the Nokia N-series [2]. However the Nokia N900 is the only mobile device released for Maemo.

The main advantage of an open source operation system on smartphones is that users can install and build software as they want without making changes in the core system. It is possible to install a terminal emulator from the official repositories and get direct access to the system. In this case users have a command line interface with root permissions. That allows full access to all files in the filesystem, even the kernel.

Maemo is still in development by strong and active community. The amount of available applications is huge and the community has created a lot packages for it. Also the system interfaces in Maemo are open-source software.

The first part of this paper focuses on special applications for Maemo. Also, possible alternatives are shown. A couple of applications are choosen to take a closer look to. These are popular or in my opinion useful applications. They can either be installed on the smartphone from a software repository or by manual placing it in the file system. Also, the package tools APT and dpkg can be used.

The second part of this paper is about developing new applications. Therefore, it is neccessary get a more detailed overview of the graphical environment and the system interfaces. A practical part contains information about the development kit and a possible way to set up a development environment. An example application is written, compiled and ran in the development environment.

The hardware used for practical testing is a Nokia N900 with Maemo Fremantle. It is shown in Figure 1. The N900 contains of sensors which all needs to be accessed over an interface.

Fig. 1. The test hardware Nokia N900 [3]

II. Software Sources

Most applications for Meamo are free software, available also as source code. Because it based on Debian, it also uses *.deb* package files to install applications. They are available through different repositories. An manual installation is also possible using APT in a terminal emulator. The recommended way is to install application using a repository.

A. Official Repository

This repository [4] contains the officially supported applications, system libraries, and their dependecies. For a basic installation, "this is all a user needs" for the normal use of the mobile device. Unfortunately, the last update to this repository was in 2011!

B. Community Repositories

The major community repository [5] is the *extra* repository. A huge amount of packages is available there. The applications are mostly stable and tested. Nokia provides the server capacities for the maemo.org community. Fresh build packages and untested applications are available in the extra-testing repository. Installing applications from this repository can be risky. Some packages may install large files in the root partition instead of the larger partition. If the root partition has no empty space the system will not work properly anymore. These applications also may not follow power-saving rules shorting the battery life-time. But these risks are reversible and should not harm the system: the malfunctioning package can still be uninstalled.

Since its very easy to set up a repository server, there are more servers available which distribute applications. Most of them provide very special applications (and their updates). Setting up an own server can be considered for the distribution of an applications if an application should not be available to the whole community.

Unlike the repositories where the applications are mostly open-source, commercial applications can be bought and installed in the Ovi Store. After the description of the Ovi Store and the pricing strategy, a few example programs will be shown.

C. Ovi Store

The *Ovi Store* [6] has recently been renamed to *Nokia Store* offers content for Symbian, MeeGo, and Maemo. The applications are commercial but there are also applications which are free to use. Its the equivalent to the *Apple App Store* [7] or *Google Play Store* [8] but with a fewer number of apps. Since Maemo is not under Nokia's development anymore and is installed only on a small number of smartphones, most proprietary popular applications are not build for Maemo. Most of the free applications in the *Ovi Store* are games and small utilitiy programs but also wallpapers, ringtones and themes.

The utility category offers the most interesting applications. On other mobile operation systems these applications are features of the core system. For Maemo they have to be installed. Also a normal users would miss these applications. *Stopwatch* and *Egg timer touch* are two examples for these applications. They are timing applications. Maemo only has an alarm clock. Another one is *U-done* a simple To-Do manager.

Connecting to a *Facebook* account in Maemo is a problem. After the connection has been successfully established, all facebook contacts will be loaded into the address book. The address book has no grouping option and is therefore flooded with contacts. *fMobi* is a more useful application to connect to facebook from the *Ovi Store* with costs around €2.

An application can easily be published in the Ovi Store using a Nokia account. On the Nokia publisher website [9] are guides available with instructions and information in several languages. All playments will be done by Nokia: they charge you on a credit card or phone bill. Having a sim card installed in the device is a requirement. This restricts the use of *Ovi Store* applications on internet tablets. In-App payments also have to be done via Nokia. Therefore, a payment API is provided. According to the *Finance and Payout FAQ* [10], Nokia will keep 30% of all payments which is exactly the same amount that Apple keeps for applications in their *App Store*. The pricing model is quite simple: there are 16 pricing levels. The prince for each level varies per country. In Germany, it ranges from free to €99.99 [11].

Maemo is not build to protect bought applications from getting copied. The applications are also installed as a debian package. Thus, an application could be copied to other smartphones which is not legally allowed but possible. Applications with In-App payments can have their own technique to check if someone bought it, by example a license server.

After these example applications from *Ovi Store* special applications will be shown. Unlike the shown applications from Ovi Store these special applications brings more functionalities that are not very common on a smartphone.

III. Special Applications

The applications described in this section are normally not installed in a default configuration. Some of them have not been stable by the time of writing. Most applications are available through the official repository or through a community repository. The following applications are choosen by its popularity. The first two are location-based applications, the second two are office applications. After that the media player will be shown.

A. HereAndNow

HereAndNow is the Nokia equivalent to *Apples AroundMe* and shows information and places near to the user's current location. In contrast to *AroundMe* and other applications with a similar idea, the *HereAndNow* version for Maemo shows only weather data and restaurants. The list of restaurants is quite short: in a test run in the center of Oldenburg, only two restaurants have shown up, but in fact there are a lot more. Other location-based serivces have more categories to locate, *HereAndNow* is not ready for productive use jet. The development needs to be continued.

B. Maps

The default application for Maps from Ovi Store is pre-installed on the N900 but not included in Maemo. It uses the *NAVTEQ* [12]. maps which are used in many navigation systems. The current position can be located by the GPS device. It works like a navigation system but without voice. It has a special drive mode which is automatically recommended

if a route longer then 50km is selected. But away from navigation, the application can also be used as an ordinary map browser.

OpenStreetMap [13] is also a map provider. The maps are available through a selection of applications [14]. For Google Maps, there is only a web-based application available [15].

It is a well-known issue that the Nokia locating server is extremly slow [16]. Using the Nokia server it takes about 5 minutes to locate the current position. The simplest solution is to use another location server, e.g. from Google [17].

C. Docs To Go

Docs to Go is installed as a 30-day trial from Dataviz [18]. A full license can be purchased for 29.99$. It can open and edit *Microsoft Word*, *Excel* and *Powerpoint* documents. The developers promises that the original formatting will never be lost. The view-only version is free of charge to use but with a notice on every use of the applocation to buy full version.

D. FreOffice

FreOffice is, like *Docs to Go*, an application to open and edit office documents powered by *KOffice*. In contrast to *Docs to Go FreOffice* is an open-source application. *FreOffice* is available through the *extra-devel* repositoy, the low-level repository under *extra-testing*. This means, the application is not even in the testing phase and might be unstable. It is recommended to not use it in a production context. The last update happend in December 2011 which means it is propably not under active development anymore.

E. Media Player

The pre-installed media player can play music, videos and internet radio Since the Nokia N900 has a large storage capacity, the media player is very important. Figure 2 shows the main menu of the media player. It can be accessed by a *Qt Mobility API* VI-B. It is very basic and works only in landscape mode. There are no settings and nothing to customize.

Fig. 2. Maemo Media Player

In the repositories are many alternatives to the pre-installed media player. For example, the *SomePlayer* [19] supports portrait mode, has a playlist management, an equalizer and more audio codecs. Figure 3 shows the playback screen. It is written in *C++* using the *Qt* library and can be controlled through a *DBus* interface. It supports bluetooth headsets and can stop when the headset or hearphone is unplugged but does not start when it is plugged in again. Also, an automatic stop timer in included. The player is optimized for music. It cannot play videos. The player can be installed through the *extra-testing* repository.

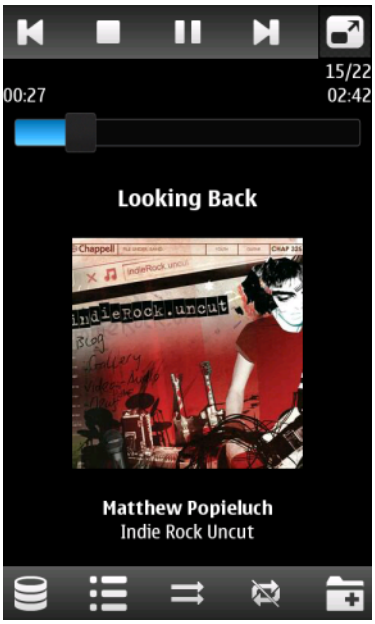


Fig. 3. SomePlayer playback screen [19]

F. Fremantle Stars

Some community projects are supported by Nokia. They are called *"Fremantle Stars"*. They should be ready-to-use by the time Fremantle was released. Some of them are and some did not finished then. There are eight applications on the list [20]:

- eCoach - is an outdoor activity recording and managing application. It records the track, speed, distance and some more values. It can be installed through the extra repository.
- FBReader - is an E-book reader for many plattforms. It is not plattform-independent and custom built for each plattform separately. The Maemo 5.0 version is stable and can be installed through a repository. The supported formats are FB2, plain text, html, rtf, chm and plucker. Though, *EPub* and *PDF* are not supported.
- liqbase - is a drawing application. The user interface is optimized for touchpad use. It is also available through the extra repository.
- Mauku - is a microblogging application. By the time of writing, it is still in the beta phase of development. It supports Twitter, Laconica and Qaiku [21]. It is based on microfeed, a command-line microblogging backend.

- NumptyPhysics - is a virtual gravity game. The user have to build ramps, arms and tackles to move two objects together.
- OMWeather - is a Weather forecast widget for the Hildon desktop. By the time Maemo Fremantle was released, it used to be the default application for weather forecasting. It is included in the Maemo core system.
- OSM2Go - is an OpenStreetMap [13] editor. OpenStreetMap is a collection of maps voluntary maintained by a huge amount of people like Wikipedia [22]. This application can edit these maps.
- Vagalume - is an application to tune in Last.fm, a portal for internet radios.

This is just a small selection of applications. There are more applications which can be used alternatively to other applications. But all applications have one common part: they all have a graphical user interface (GUI). Because building a seperate user interface for each application increases the programming time and would propably not have a consistent design. Therefore, a Framework is used as a part of Maemo: the Hildon Framework.

IV. Hildon Framework

The Hildon Framework is an application framework for mobile devices. It is part of Gnome. The user interface is optimized for touchscreen devices to interact using fingers or a pen with a mobile device. The Hildon Framework consists of three parts:

The *Hildon Program Manager* provides a user interface to Debian's package management tool *APT* and *dpkg*. It has limited access to the system for protecting a user from destroying the operation system. Figure 4 shows the categories in which the applications are differentiated. System files and core libraries cannot be removed, but updated. A user can gain full power using aptitude, apt, synaptics, etc.

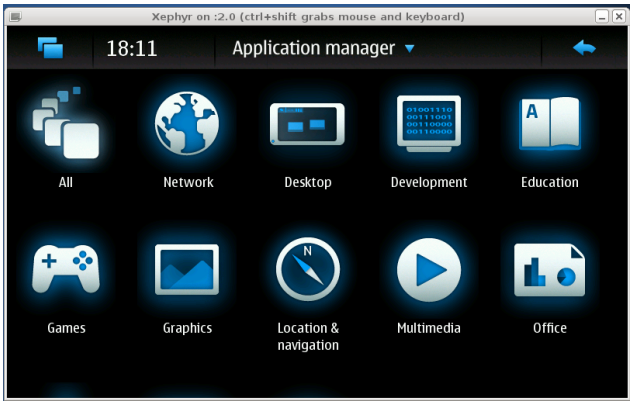


Fig. 4. Hildon Program Manager

The *Hildon Control Panel* is a powerful tool to set up the system, communcation and personalization. In the control panal, a theme can be selected changing the style of the desktop and all applications.

The *Hildon Desktop* is the most visible part in Maemo. It provides the desktop, the taskbar, switching betweem applications and contols applets and widgets on the desktop. Applications for the *Hildon Desktop* should follow a style guide [23]. The style guide shows describes the minimal hardware the application should support. It also shows how a user normally acts with a Maemo based smartphone. There are some differences between the described hardware keys and the keys that actually exists on a N900. The home key and the menu key, do not exist. The other chapters aim at the design of applications. The basic components, views, menus, toolbars, dialogs and input methods are very important for the usability. A user should be able to interact instnictively with the application just like applications he/she already knows. This makes visual consistence to a major concern. To reach it, all application should base on the same GUI toolkit or at least a compatible one. In case of Hildon and Gnome; the toolkit used is *GTK+*.

V. GTK+ and Qt

GTK+ is one of the most popular GUI toolkits. GTK+ GUIs can be developed using a *GUI builder* like *Glade*. Alternatively the GUI can be developed manually. It is written in *C*, but many other popular programming languages have bindings to it. *Java* has *java-gnome*, *Python PyGtk* or *PyGObject*. Also, *Mono* with *Gtk#* and *Ruby* with *ruby-gtk2* have bindings.

Another very popular toolkit is *Qt*. In contrast to *GTK+*, *Qt* is not only a GUI toolkit but has much more functions like modules for multimedia, network and database use. The GUIs can also be created with an GUI builder. Creating the GUI manually seems to be harder than in *GTK+*. Since Trolltech, the company behind *Qt* is part of Nokia, [24] the *Qt* libraries are also supported. It is written in *C++* but has bindings like *Gtk+* to other languages.

All these languages are also supported by Maemo [25], some require to install runtime environments and other dependecies. On the one, hand programmers can choose a language they like and which is easy to read and to program. In the case of an easy to read and program language, Java or Mono seem to be good options. On the other hand using native libraries and toolkits which are already installed saves storage space in the root partition on the smartphone and also prevents bugs caused by the binding.

Considering the last point, *C* with *Gtk+* is the optimal option. It is also the only official programming language for Maemo. All neccessary libraries are already installed on Maemo, the compiler is also integrated into the *Software Development Kit (SDK)* [26] and the use of *Gtk+* is easier than using *Qt*.

The next step is to get more information about the system interfaces. Applications need to get access to sensors, actors and other applications.

VI. System Interfaces

Since Maemo Fremantle, the system interfaces are provided by *Qt Mobility 1.2*. Nokia uses *Qt* as common application

framework. Qt Mobility is a collection of currently 14 APIs [27]. In some APIs like the Sensor API there are notes for Maemo 5 on the N900 which decribes the use of the API on the N900.

A. Sensors

The Sensor API gives access to many sensors which can be used to improve the interaction with the smartphone. Most sensors are not frequently used: They are available but programming applications which detect movement and improve user interactions them take much time. To detect movement the accelerometer and the rotation sensors are important as well as the orientation sensor.

The accelerometer returns three values for the acceleration along the three axis. The rotation sensor measures the rotation of the smartphone. They three axes also be called roll, pitch and yaw, the same axes like in an airplane control. Figure 5 shows an overview of accelerometer and rotation sensor. This sensor is the most important sensor for implementing a gesture detection application.

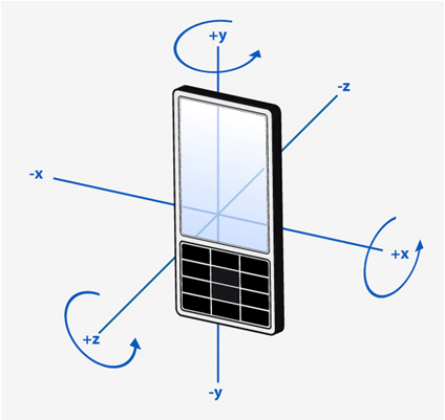


Fig. 5. Axes of accelerometer and rotation sensor (source: [27])

In contrast to the rotation sensor which can only detect active movement, the orientation sensor determines which side of the smartphone is pointing up. The six possible cases are represented by an enumeration: top, bottom, left side, right side, front and back. Four possible orientations are shown in figure 6 It would make sense to rotate the desktop if the sensor detects that the device is bottom up but is not implemented at the time of writing.

Another interestion sensor is the proximity sensor. It can detect whether an object is close to the smartphone or not. In the N900, the proximity sensor is placed next to the front camera. The distance the sensor detects a close object is not clearly defined. It is device-specific and varies for different models.

The magnetic sensor measures the magnetic field density along the three axes. The compass depends on this sensor but only returns the azimuth (the clockwise angle to the south pole of a magnetic field. The sensor can be used in physics applications. This sensor needs to be calibrated. This is simple

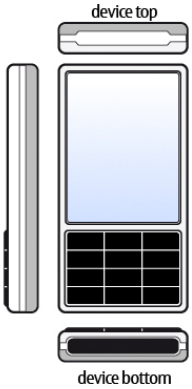


Fig. 6. Possible orientations (source: [27])

a null value measurement to compensate disturbing influences. For the compass, the sensor is pre-calibrated.

The last sensor is a tab sensor. It can detect a double tab and the tap direction on the three axis. The sensor can also be used to improve the concurrent interaction. The smartphone can be controled by just tabbing it. By example to deny a phone call or to switch to silent mode. It is possible to access the smartphone in the pocket by knocking on it. The sensor also detects the direction of the tap.

These are the sensors that can be used by Maemo applications. The camera is also a sensor to get information from next to the smartphone.

B. Multimedia and Camera

With the multimedia API, applications can have access to the media player, the audio recorder and the camera. The media player class can be used to add files to the playlist, control the volume and other elements from the media player. Audio can be captured and be saved in a raw file. Encoding is not supported and needs to be done by another library.

The camera can capture photos or videos. The camera focus has different modes to work with. These are automatic focus, continuous focus, infinity focus and a point mode which is useful for face recognition. The library also controls auto-exposure and white balance.

C. Connectivity

Another important API is the connectivity API. It is responsible to manage bluetooth connections and *Near Field Communication (NFC)* [28].

Bluetooth is a very powerful standard that many devices support. Having a full *Bluetooth* stack available in a smartphone extends the possibilities of interaction. An application can act by example as a remote control for televisions or bluray players but can also be controlled by *Bluetooth* remote controls.

The *NFC* is used to communicate with other *NFC* devices within a range of about 10 cm. For example *NFC* is used for the micropayment service *girogo* and the *Touch&Travel* system of *Deutsche Bahn*.

D. Contacts, Messaging and Organizer

Since Maemos integrated contact management is not so powerful, it can easily be extended by using the Contacts API. The contacts in Maemo are very detailed. A contact entry has many pre-defined fields, a thumbnail and a global unique identifier which allows to synchronize a contact with other data stores. Furthermore, a contact can be linked into a one-to-one relationship to another contact. This is a very useful feature but is not used in the build-in contact application. Another feature of the contact management API is to observe a contact. This means a change in a contact causes a signal which can be used to notify other contact applications. The contacts can be accessed synchronously and asynchronously. Normally programmers should use asynchronous access which does not block other applications from accessing contacts. But the synchronous way can also be used over a manager class.

The messaging API provides an interface to the messages stored in the smartphone and offers the ability to send messages, either through SMS, MMS, email or instant messageging.

This API provides a basic interface to calendar and events. In contrast to the contacts, the organizer can only be accessed synchronously.

E. Location

Another interesting interface can be used to access GPS data. It provides information about satellites in range of the current position, gives information about a place next to the current position and the position coordinates themselves.

F. System Information

This part is used to provide system information. It does not have write access to it. The profile information contains data about the ringtone volume, the vibration and the voice volume. For the battery is the nominal capacity, the remaining capacity and the current flow available. Also network information is available. It shows the network type and state, also the IP and MAC address.

There are more information which are simply string values for device, display, screensaver and storage state information. Unlike these interfaces which are accessible by the *Qt Mobility APIs* the rainbow LED and the vibration only be controlled using a *DBus* [29] interface.

G. Rainbow LED and Vibration

The Rainbow LED is a very special feature of the N900. It is placed in the bottom left corner and can adapt its color. *Mode Control Entity (MCE)* [30] is the program in Meamo that controls the LED. It does not have an API but it can be accessed by DBus communication.

The Vibration or rumble function can either be used by the Feedback API from the Qt Mobility collection or by the MCE over DBus. [31]

Before a new application for Maemo can be developed, a development and testing environment needs to be set up. Therefore, the *SDK* is provided by Nokia but it does not seems to be maintained anymore.

VII. Maemo SDK

The Maemo SDK is basically a sandboxed environment based on scatchbox. It supports the x86 and the ARMEL plattform. Also, all neccessary tools to build Debian packages are included. Most applications developed in this environment will have a GUI. Therefore, a GUI testing environment is neccessary. It is possible to install the *Hildon Framework* (see section IV) into the sandbox. This requires an own *XServer* instance to run it. A little tool called *xephyr* can create a pseudo XServer in a window that can be used to display the desktop.

The simplest solution for all these issues is to setup a virtual 32-Bit Debian based system. To develop the example application, a virtual *Debian Squeeze* in installed using *Oracle VirtualBox* [32]. After the installation of the operation system, scratchbox can be installed from the package repositories and the install scripts are working too. [26] shows a guide how to install the development environment.

An “Hello World” example can be found on [33]. It runs in the created environment. Figure 7 shows an example application based on the *hello world* example. These steps are neccessary to run it in the testing environment:

```
#Start the pseudo XServer
Xephyr :2 -host-cursor -screen 800x480x16 \
-dpi 96 -ac &
#change the users GID temporary to sbox
newgrp sbox
#start the scratchbox
/scratchbox/login
#compile the example gtk_helloworld-1.c
gcc -Wall -g gtk_helloworld-1.c \
`pkg-config --cflags --libs gtk+-2.0` -o
#start the hildon framework
af-sb-init.sh start
#run example application
run-standalone.sh ./gtk_helloworld-1
```

Using all these information about the SDK, the system interfaces and the *Hildon Framework* able to create applications using the Hildon API reference [34]. Writing useful applications that interact with the hardware requires not just the GUI elements but also knowledge of the system interfaces.

VIII. Conclusion

The system interfaces allows to create a lot of very powerful applications. They can improve the normal use of the smartphone. Also, the data integrity of the data between the smartphone and other communication devices like a computer can be improved. It allows to develop new ways of interaction with other dynamic devices. Maemo and the hardware on which it runs on is complex enough to do this, but the applications are not developed at the time of writing.

On the one hand the system interfaces are very powerful but on the other hand very complex. The recommended programming language is C which is a difficult to read and program for most smartphone developers. There is only a small amount

Fig. 7. Example application

of C developers for mobile devices in comparison to java developers. In comparison to Android or IOS programming the developing time is longer and more expensive, because the programmers need have deeper understanding of the system. Also, unlike in iOS and Android, the application framework is not so much abstracted.

Without commercial-driven development, Maemo will be out-of-date quite near in the future. A real passionate users could write applications for their own, but it will not fit the interest of the average user. The average user will have the mobile device working without much effort to set up the system. The Maemo core system does not contain all functions users need. Other plattform like Android and IOS have better default applications. There is no need to install applications to manage their contacts or their organizer. But Maemo may be a good plattform for big companies if they want to have a full integrated groupware solution also on mobile devices. They can spend the effort to write applications perfectly for their use.

Maemo becomes more and more complicated for normal users since most applications are not under active development anymore. At the time of writing the maemo.org website has not been available for several days. Maemo is no product mass marked, only for passionate users. However, the Maemo community seems to become more and more quiet and the next year will decide if the Maemo development will be continued or not.

REFERENCES

[1] Nokia, "The maemo platform," 2012, [accessed 11.12.2012]. [Online]. Available: http://www.developer.nokia.com/Devices/Maemo/

[2] ——, "Device specification," 2012, [accessed 01.02.2013]. [Online]. Available: http://www.developer.nokia.com/Devices/Device_specifications/?filter2=maemo

[3] Golem.de, "Neues community-update für nokia n900," 2012, [accessed 11.12.2012]. [Online]. Available: http://www.golem.de/1109/86319.html

[4] Nokia, "Official repository," 2012, [accessed 01.02.2013]. [Online]. Available: http://downloads.maemo.nokia.com

[5] M. Commutity", "Community repository," 2012, [accessed 01.02.2013]. [Online]. Available: http://repository.maemo.org/

[6] Nokia, "Ovi store," 2012, [accessed 11.12.2012]. [Online]. Available: http://store.ovi.com/

[7] A. Inc.", "Iphone app store," 2012, [accessed 01.02.2013]. [Online]. Available: http://www.apple.com/de/iphone/from-the-app-store/

[8] Google, "Google play store," 2012, [accessed 01.02.2013]. [Online]. Available: https://play.google.com/store?hl=de

[9] Nokia, "Publisher guides," 2012, [accessed 01.02.2013]. [Online]. Available: http://support.publish.nokia.com/?page_id=4547

[10] ——, "Finance and payout," 2012, [accessed 23.01.2013]. [Online]. Available: http://support.publish.nokia.com/?cat=6&topic=57

[11] ——, "Billing matrix," 2012, [accessed 23.01.2013]. [Online]. Available: https://admin.support.publish.nokia.com/wp-content/uploads/2012/09/Billing_Matrix_Sep6_2012.pdf

[12] NAVTEQ, "Navteq maps," 2012, [accessed 17.01.2013]. [Online]. Available: http://www.navteq.com/deutsch/company.htm

[13] OpenStreetMap, "Openstreetmap," 2012, [accessed 01.02.2013]. [Online]. Available: http://www.openstreetmap.de/

[14] ——, "Software/maemo," 2012, [accessed 17.01.2013]. [Online]. Available: http://wiki.openstreetmap.org/wiki/Software/Maemo

[15] T. Waelti", "Maemaps," 2012, [accessed 17.01.2013]. [Online]. Available: http://tomch.com/maemaps.html

[16] T. N. Blog", "Gps lock taking too long..." 2012, [accessed 01.02.2013]. [Online]. Available: http://thenokiablog.com/2009/12/16/google-location-server-supl/

[17] Google, "Location server," 2012, [accessed 01.02.2013]. [Online]. Available: supl.gogle.com

[18] Dataviz, "Documents to go for maemo," 2012, [accessed 17.01.2013]. [Online]. Available: http://www.dataviz.com/products/documentstogo/maemo/

[19] Mustali, "Someplayer – an awesome audio player for the n900," 2012, [accessed 01.02.2013]. [Online]. Available: http://myn900.wordpress.com/2011/03/28/someplayer-an-awesome-audio-player-for-the-n900/

[20] M. Commutity", "Fremantle stars," 2012, [accessed 11.12.2012]. [Online]. Available: http://wiki.maemo.org/Fremantle_Stars

[21] Mauku, "Multiple services," 2012, [accessed 26.01.2013]. [Online]. Available: http://mauku.innologies.com/features/multiple

[22] Wikipedia, "Wikipedia," 2012, [accessed 01.02.2013]. [Online]. Available: http://en.wikipedia.org/wiki/Wikipedia

[23] Nokia, "Hildon user interface style guide," 2012, [accessed 12.01.2013]. [Online]. Available: http://maemo.org/forrest-images/pdf/UI_Style_Guide_Summary_2.0.pdf

[24] Golem.de, "Nokia kauft trolltech," 2012, [accessed 01.02.2013]. [Online]. Available: http://www.golem.de/0801/57278.html

[25] M. Commutity", "Programming languages," 2012, [accessed 11.12.2012]. [Online]. Available: http://maemo.org/development/documentation/programming_languages/

[26] ——, "Maemo sdk," 2012, [accessed 16.01.2013]. [Online]. Available: http://wiki.maemo.org/Documentation/Maemo_5_Final_SDK_Installation

[27] Nokia, "Qt mobility project reference documentation," 2011, [accessed 16.01.2013]. [Online]. Available: http://doc.qt.digia.com/qtmobility/index.html

[28] F. M. (Ed.)", "Adjunct proceedings," 2008.

[29] freedesktop.org, "Dbus," 2012, [accessed 01.02.2013]. [Online]. Available: http://www.freedesktop.org/wiki/Software/dbus

[30] M. Commutity", "Mode control entity," 2012, [accessed 01.02.2013]. [Online]. Available: http://wiki.maemo.org/Documentation/Maemo_5_Developer_Guide/Architecture/System_Software#Mode_Control_Entity_.28MCE.29

[31] Nokia, "Using mce interface for vibration activation in maemo 5," 2008, [accessed 17.01.2013]. [Online]. Available: http://www.developer.nokia.com/Community/Wiki/Using_MCE_interface_for_vibration_activation_in_Maemo_5

[32] Oracle, "Virtualbox," 2012, [accessed 01.02.2013]. [Online]. Available: https://www.virtualbox.org/

[33] M. Commutity", "Hello world application," 2012, [accessed 16.01.2013]. [Online]. Available: http://wiki.maemo.org/Documentation/Maemo_5_Developer_Guide/Development_Environment/Maemo_SDK#Writing_GUI_Hello_World

[34] ——, "Hildon api," 2012, [accessed 16.01.2013]. [Online]. Available: http://maemo.org/api_refs/5.0/beta/hildon/index.html