Anwendungsentwicklung für mobile Endgeräte -

Stand der Technik, Entwicklungsumgebungen und Trends

Gabriel Schreiber

Universität Konstanz

www.bachelorarbeit.gschreiber.com

Danksagung

Diese Bachelorarbeit entstand aus der Faszination heraus, mich mit kleinen, mobilen, tragbaren „Funkspielzeugen“ zu befassen. Mein besonderer Dank an Herrn Prof. Wolfgang Pree, der mir im Zuge dieser Bachelorarbeit die Möglichkeit gab, „das nützliche mit dem angenehmen zu verbinden“ und daraus eine Bachelorarbeit anzufertigen. Vielen Dank für die Bereitschaft, mich trotz des Wechsels nach Salzburg aus der Ferne zu begleiten. Mein Dank geht auch Herrn Prof. Reiterer, der zustimmte, die Arbeit als Zweitkorrektor vor Ort zu betreuen.

Besonderer Dank an die Mitarbeiter des Lehrstuhls für Software Engineering, insbesondere Sebastian Fischmeister für die Bereitschaft, mich trotz seines Weggehens nach Salzburg weiterhin zu betreuen und zu motivieren sowie an Matthias Lampe, für seine ausführlichen Erklärungen.

Vielen Dank an die Dozenten und Mitarbeiter der Universität, von denen ich im laufe des Studiums viel Wissen vermittelt bekam. Danke an meine Mitstudenten, besonders Birgit Nitschmann, ohne die das Studieren nur halb so viel Spaß gemacht hätte.

Nicht zuletzt gilt mein ganz besonderer Dank Katharina Schulz die mir die ganze Zeit über mit intellektueller und vor allem emotionaler Unterstützung zur Seite stand und mir den Rücken für mein Studium frei hielt.

Hinweis

Zusammenfassung

Um einen Überblick über bestehende Betriebssysteme für mobile Endgeräte zu bekommen, werden die drei Betriebssysteme Windows CE, PalmOS und Symbian anhand ihrer Eigenschaften und Charakteristika beschrieben. Darauf folgend wird MExE als eine realisierte Serviceplattform des 3GPP vorgestellt.

MExE ist unterteilt in drei Classmarks, die näher beschrieben werden. Dabei wird auf WAP (Classmark 1), PersonalJava (Classmark 2) und J2ME mit CLDC und MIDP (Classmark 3) näher eingegangen. Für jedes dieser Classmarks werden Entwicklungsumgebungen und Endgeräte vorgestellt. Die Problematik der Softwareverteilung und Installation wird in einem weiteren Kapitel behandelt.

Im praktischen Teil wird eine Anwendung zur Zeiterfassung namens *myTime* für alle drei Classmarks implementiert und beschrieben. Dabei wird jeweils an einem Beispiel mit einer Entwicklungsumgebung und einem Endgerät das genaue Vorgehen aufgezeigt und auf Besonderheiten und Beschränkungen hingewiesen.

Abstract

In order to receive an overview of existing operating systems for mobile devices, we first look at the three operating systems Windows CE, PalmOS and Symbian. They are described on the basis of their characteristics and features. Whereupon following MExE is introduced as an implemented service platform of the 3GPP.

MExE is divided into three Classmarks, which are described more in detail. WAP (Classmark 1), PersonalJava (Classmark 2) and J2ME with CLDC and MIDP (Classmark 3) is specified in greater detail. For each of these Classmarks development environments and terminals are introduced. The difficulty of software distribution and installation is treated in a further section.

In the practical part of this work an application for time recording named *myTime* is implemented and described for all three Classmarks. With an example, using a development environment and a device, the exact procedure of implementation is pointed out in each case. Special features and limitations of each implementation are presented.

1 Betriebssysteme mobiler Endgeräte

Aufgrund des aktuellen Verbreitungsgrades (siehe Abschnitt 1.4) relevanter Betriebssysteme werden die wichtigsten (WindowsCE, PalmOS und EPOC) jeweils nach folgenden Kriterien untersucht:

- **Betriebssystemcharakteristika:** Benötigter Speicherplatz, Minimalanforderung an Prozessorgeschwindigkeit, Stabilität, Netzverbindungen, Multitaskingfähigkeit, Boot-Vorgang
- **Unterstützung eines GUI-Standards**
- **Application Programming Interface (API)**
- **Kompatibilität:** Wie gut können Daten und Anwendungen mit anderen Systemen ausgetauscht werden, wie weit werden Standards direkt oder indirekt unterstützt
- **Hardwarecharakteristika:** Größe (Gewichte, Abmessungen), Speicherbereitstellung (wie viel Speicher kann für Daten genutzt werden), Hardwareanforderungen, vorhandene Hardware auf dem Markt.
- **Entwicklungswerkzeuge:** Wie können Applikationen entwickelt werden

1.1 Microsoft Windows CE

Durch die Einführung des offensichtlich gut angenommenem iPaq – dieser verhalf Compaq zu einer Steigerungsrate von 800% im Absatz von PDAs – ist die Vorherrschaft des PalmOS und den zugehörigen Geräten am bröckeln und Microsoft mit WindowsCE ist am aufholen.[I] [dataq]

1.1.1 Betriebssystemcharakteristika

Windows CE gibt es in drei Versionen:[ceapp]

- **Windows CE 3.0** ist für den Pocket PC vorgesehen. Es ist Windows auf einem PC ähnlich.
- **Windows CE 2.12** wird hauptsächlich von Herstellern integrierter Systeme genutzt, die den *Microsoft Plattform Builder* verwenden, um mit ihm ihre Systeme zu realisieren
- **Windows CE 2.21** Diese Version von CE wird in Handheld PCs und in Palm Size PCs verwendet.

Weitere Angaben zu Speicherplatz, Minimalanforderung an Prozessorgeschwindigkeit, Stabilität und Bootvorgang konnten nicht herausgefunden werden. Netzverbindungen und Multitaskingfähigkeit sind analog zu den Windowsversionen für PCs gegeben.

Das Betriebssystem Windows CE zeichnet sich durch folgende Besonderheiten aus: [ceapp]

[I] Stand September 2001

- Zu Windows NT und Windows 2000 kompatible API
- Multitasking und Synchronisation wird vom Betriebssystem unterstützt
- Filesystemzugriff
- TCP/IP-Stapel mit Funktionen, die HTTP- und Socketverbindungen zulassen
- Zugriff auf Windows NT und Windows 2000 Netzwerkfunktionen
- Kommunikation über die serielle Schnittstelle
- ADOCE (ActiveX Data Objects für Windows CE)
- COM-Unterstützung
- DCOM-Unterstützung
- ActiveSync für Synchronisation mit PC
- in Zukunft voraussichtlich .NET-Unterstützung

1.1.2 GUI

Abbildung 1: WindowsCE – GUI
Quelle: Compaq

Das GUI (Graphical User Interface) ist dem von Windows sehr ähnlich. Funktionen wie Eingabe mittels Zeigestift und virtueller Tastatur werden unterstützt. Die virtuelle Tastatur wird bei Bedarf eingeblendet. Mit dem Zeigestift können darauf Buchstaben ausgewählt und so in die Anwendung eingefügt werden.

Auch die Taskleiste mit dem Windowssymbol ist auf Windows CE vorhanden. Laufende Applikationen werden in der Taskleiste dargestellt, sobald sie in den Hintergrund geschickt werden.

Um Anwendungen zu starten oder Aktionen auszulösen, werden mit dem Zeigestift Menüs ausgeklappt und die darauf vorhandenen Menüpunkte ausgewählt. Auch eine direkte Anwahl der Icons über die Desktopmetapher ist mittels Zeigestift möglich.

1.1.3 API

Das API ist bis auf wenige PDA-spezifische Zusätze mit dem von Windows für Desktop PCs identisch. Microsoft hat Werkzeuge (JUMP) angekündigt, die es erlauben, Java in C# zu konvertieren. WindowsCE wird in die .NET Strategie von Microsoft einbezogen und alle zukünftigen Entwicklungen werden sich auf .NET und nicht auf Java konzentrieren. [micro]

Für WindowsCE existiert eine PersonalJava Umgebung, die von Sun angeboten wird[pjae].

1.1.4 Kompatibilität

In der Vergangenheit mussten WindowsCE Anwendungen für den jeweiligen Prozessor angepasst bzw. kompiliert werden. Diese Prozessoren waren unter anderem MIPS und SH3. Die neueren Versionen der CE-Geräte unterstützen einen prozessorneutralen Maschinencode namens CEF (Common Executable Format). Ist eine Applikation für eine CEF-kompatible Plattform geschrieben, so wird auf der Plattform der CEF-Code in Maschinencode übersetzt und ausgeführt. Der übersetzte Code kann gespeichert oder bei jedem Programmstart neu übersetzt werden. Diese Kompatibilität geht auf Kosten der Leistung. CEF-Code ist etwa 20% langsamer als prozessorspezifischer Maschinencode. Es gibt für CEF-Code keine Just-in-Time-Compiler (JIT) wie für Java. [ceapp]

Kompatibilität der Daten zu gängigen Standardprogrammen wie Word oder Inhalten aus dem Web wird mit den in das System integrierten Programmen wie Pocket Word, und Pocket Internet Explorer erreicht.

1.1.5 Hardware

Windows CE ist auf die Geräte Pocket PC, Handheld PC und Palm Size PC ausgelegt, die Microsoft[micro] wie folgt charakterisiert:

Pocket PC - Der Pocket PC hat keine Tastatur, erkennt aber handgeschriebene Zeichen oder erhält diese von einer virtuellen Tastatur, die bei Bedarf eingeblendet wird. Pocket PCs unterstützen entweder Typ 1 oder Typ 2 CompactFlash Karten. Die Steckplätze können entweder mit Festspeicher ausgestattet werden oder ein Winchesterlaufwerk aufnehmen. Es können Datenträger ausgetauscht werden, oder Peripherie wie Mobiltelefone, Modems, Barcodeleser oder Kameras angeschlossen werden. Ein Pocket PC hat eine Anzeige von 320x240 Pixel (1/4 VGA) und besitzt einige im System integrierte Anwendungen, wie beispielsweise einen Web-Browser oder den Windows Media Player. Pocket PCs basieren auf den Prozessoren MIPS und Intel StrongARM. [ceapp]

Handheld PC – Der Handheld PC unterscheidet sich vom Pocket PC hauptsächlich in der Größe und ist zudem mit einer Tastatur ausgestattet. Handhelds unterstützen meist eine PCMCIA-Schnittstelle und haben oft ein eingebautes Modem. Mausunterstützung wird dadurch simuliert, dass die Geräte einen Touchscreen besitzen oder ein Trackball in der Tastatur integriert ist. Ein Handheld PC besitzt entweder eine halb-VGA (640 x 240

Pixel) oder eine VGA (640x480 Pixel / 800x600 Pixel) Anzeige. 32 MB RAM sind mindestens erforderlich. [ceapp]

Palm Size PC – Dieser wurde weitestgehend vom Pocket PC ersetzt. Er besitzt ein GUI, das dem der Windows Desktopversion ähnelt.

WindowsCE findet man in integrierten Systemen wie den oben beschriebenen, der X-Box, WebTV und dem eBook Reader. Entwickler können auf Basis des Microsoft Platform Builders integrierte Umgebungen entwickeln, auf denen WindowsCE lauffähig ist. [ceapp]

1.1.6 Entwicklungswerkzeuge

Die WindowsCE API steht dem Entwickler in C zur Verfügung. Anfangs wurde in die Entwicklungsumgebung Visual C++ von Microsoft Erweiterungen eingefügt, die Windows CE-spezifische Funktionen zur Verfügung stellten. Der Nachteil hierbei war, dass die Funktionen, die nicht auf Windows CE verwendet werden konnten, immer noch sichtbar waren. Es wurde schließlich von Microsoft eine neue Entwicklungsumgebung mit dem Namen *embedded Visual C++* auf den Markt gebracht, die vollständig auf die Bedürfnisse der Entwicklung für CE-Betriebssysteme angepasst ist. [ceapp]

Die meisten WindowsCE-SDKs stellen Emulatoren zur Verfügung, so dass Anwendungen in diesen getestet werden können, bevor sie auf das Gerät übertragen werden. Dies spart Zeit, ist aber kein Ersatz für Tests auf dem Zielgerät.

1.1.7 Softwareunterstützung

Kompatibilität der Daten zu gängigen Standardprogrammen wie Word, Powerpoint und Excel oder Inhalten aus dem Web wird mit den in das System integrierten Programmen wie Pocket Office, und Pocket Internet Explorer erreicht.

Über diverse Schnittstellen wie Infrarot, CompactFlash Typ I und II, USB und serielle Schnittstelle lassen sich Daten zwischen Desktop PC und Windows CE Geräten synchronisieren.

1.2 PalmOS

1996 wurde das erste PalmOS[palm] (Palm Operating System) für den Pilot 1000 herausgegeben[palmab] [palmsto]. Seitdem haben Geräte, die auf dem PalmOS basieren, den Markt für sogenannte PDAs (Personal Digital Assistant) dominiert. Momentan zeichnet sich eine Trendwende ab. Andere Hersteller mit Betriebssystemen, wie die in diesem Kapitel repräsentativ aufgeführten, haben den Markt für mobile Endgeräte entdeckt und ziehen mit Ihren Produkten nach.

1.2.1 Betriebssystemcharakteristika

Das PalmOS versucht nicht, wie Windows CE, eine PC-Umgebung auf einem PDA abzubilden, sondern orientiert sich von vornherein an den Anforderungen, die an Kleinstgeräte gestellt werden. Diese sind Daten an jedem Ort schnell aufnehmen und wiedergeben zu

können. So hat das PalmOS den Anspruch, ein „Sattelitensystem“ zu sein, das einen Ausschnitt aus den Daten eines Desktopsystems mobil und sofort darstellt.

Die vier Applikationen Terminplaner, Adressbuch, Zu-Erledigen-Liste und Notizblock sind als fester Bestandteil im Betriebsystem integriert.

Palmprogramme sind „eventdriven“ und können vom PalmOS jederzeit unterbrochen oder für eine bestimmte Aufgabe aufgerufen werden. Die meisten Programme für den Palm sind in der Programmiersprache C oder C++ geschrieben. Dafür gibt es diverse Entwicklungsumgebungen.

1.2.2 GUI

Abbildung 2: Palm OS - GUI

Das GUI des PalmOS ist auf mit Zeigestift bedienbaren Elementen ausgelegt. Die Eingabe erfolgt über eine palmspezifische Handschrift in einem eigens dafür vorgesehenen Bereich oder über eine virtuelle Tastatur. Zum Umschalten zwischen den wichtigsten Applikationen dienen vier Knöpfe unter der Anzeige.

1.2.3 API

Das API des PalmOS konzentriert sich auf die folgenden Bereiche: User Interface, System Management, Kommunikationsfunktionen. Es besitzt umfangreiche Bibliotheken mit Funktionalitäten für Internet, drahtloses Internet, SMS, HotSync, Datenbank-Synchronisation, Infrarot, TCP/IP und Telephonie.

1.2.4 Kompatibilität

Kompatibilität oder leichte Portierbarkeit zu vorhandener Software, wie es beispielsweise bei Windows CE der Fall ist, ist beim PalmOS nicht gegeben. Das PalmOS ist auf Synchronisation mit dem Desktop ausgelegt. Es bietet eine Reihe von Anwendungen, die

die Aufgabe der Synchronisation übernehmen. Anwendungen werden jedoch nicht nachgebildet, wie dies beispielsweise bei Poket Word für WindowsCE der Fall ist.

Für PalmOS 4.0 wurde *WebClipping*[palmmob] für die Nutzung des Web entwickelt. Der Benutzer kann damit allerdings nicht wie gewohnt im Web surfen, sondern nur in speziellen Angeboten, die in einer vereinfachten HTML-Syntax erstellt wurden. Für jedes WebClipping-Angebot ist eine entsprechende Zusatzdatei vorhanden, die vorher besorgt werden muss. Es gibt in Europa allerdings nur wenige Angebote. In den U.S.A. ist das Angebot für *WebClipping* vielfältiger.

Für das PalmOS sind inzwischen Webbrowser verfügbar, mit denen der Anwender auch normale Webangebote aufrufen kann. Diese werden von Fremdherstellern und nicht direkt von Palm[palm] angeboten. Als Webbrowser stehen beispielsweise zur Verfügung:

1. *Blazer* von Bluelark unterstützt HTML, WAP (WML/HDML), und cHTML, das die Basis von i-mode in Japan bildet. *Balzer* ist farb- und grafikfähig. *Blazer* benutzt einen Proxy des Anbieters um, SSL-verschlüsselte Webseiten anzuzeigen. Dabei werden die Daten auf der Strecke zwischen dem Palm und dem Proxy ECC-verschlüsselt. *Blazer* unterstützt Cookies. Einmal geladene Seiten können später offline aus dem Cachespeicher angesehen werden. *Blazer* benötigt Palm OS 3.1 oder eine neuere Version und mind. 200 kB RAM.[blue]
2. *Browse-It* bietet die Möglichkeit, Webseiten über den PC zu laden und diese als *Snapshots* darzustellen. *Browse-It* unterstützt HTML und Bookmarks. Cookies werden nicht unterstützt. *Browse-It* benötigt mind. 200 KB RAM, zusätzliches RAM wird für die Webseiten benötigt. *Browse-It* wurde zum 2.1.2001 eingestellt. Diese Arbeit entstand in Ihrem Kern jedoch vor diesem Datum und deshalb wurde *Browse-It* hier noch erwähnt.[brow]
3. *EudoraWeb* stellt keine Grafiken dar. Der Browser unterstützt kein WAP. Er unterstützt Cookies und bietet eingeschränkte Frameunterstützung. Es lassen sich Bookmarks verwalten. Zusätzlich lassen sich Bookmarks zwischen Palm und Webbrowser synchronisieren.[eudo]

Der Schwerpunkt der Anwendungen für das PalmOS liegt auf der Synchronisation der Daten mit gängigen Standardprogrammen, wie etwa Microsoft Outlook, über den PC. Es gibt zwar Konverter, mit denen sich Worddokumente für das PalmOS konvertieren lassen, aber die hauptsächlichen Anwendungen des PalmOS konzentrieren sich auf Terminverwaltung, Adressverwaltung und kurzen Notizen, nicht jedoch auf Bearbeitung von langen Texten.

1.2.5 Hardware

Das PalmOS ist auf minimale Hardware ausgelegt. Der kleinste derzeit verfügbare Palm misst 113 x 79 x 10 mm und wiegt 113 Gramm. Er ist mit dem Prozessor MC68EZ328 (16 MHz) von Motorola ausgestattet. Die durchschnittliche Laufzeit der Batterien beträgt etwa 2 Monate.

Das PalmOS ist auf ein GUI ausgerichtet, welches einfach und schnell zu bedienen ist. Die Synchronisation mit dem PC ist eine wichtige Aufgabe für das PalmOS und wird dementsprechend gut unterstützt.

Die Anzeige bietet 160 x 160 Pixel. 36KB dynamisch verwalteter Speicher stehen zur Verfügung. Dieser ist auf max. 8 MB erweiterbar.

Folgende Geräte gibt es momentan für das PalmOS auf dem Markt[II] : Palm m505, Palm m500, Palm m105, Palm m100, Palm IIIc, Palm IIIxe, Palm IIIe, Palm Vx, Handspring Visor, Handspring Visor Deluxe, Platinum Handspring, Visor Prism, Handspring Visor Edge, HandEra 330, PalmPix, Seiko SmartPad und der Sony Cliè. Die meisten von Ihnen basieren auf einem Prozessor von Motorola.

Die Geräte sind teils per MultiMediaCard, CompactFlashCard und einige herstellerabhängige Karten erweiterbar, bzw. an die Peripherie anschließbar. Ein GPS Empfänger kann beispielsweise über einen CompactFlashCard Einschub nachgerüstet werden. Diverse Zusatzhardware wie einsteckbare Kameras, Bluetoothadapter, faltbare Tastaturen etc. ist für den Palm verfügbar.

1.2.6 Entwicklungswerkzeuge

Dies PalmOS ist eine populäre Entwicklungsplattform, für die es, verglichen mit den anderen Systemen, mehr Anwendungen gibt.

Das gilt auch für die PalmOS Entwicklungsumgebungen. Die gängigsten Sprachen für die Plattform sind C und C++. CodeWarrior[palmcw] ist eine kommerzielle IDE, die sowohl für Mac- als auch für Windows-Plattformen verfügbar ist. Auch unter der GPL-Lizenz (General Public License) sind Entwicklungsumgebungen verfügbar.

Außer den G-Entwicklungsumgebungen gibt es eine Reihe anderer Entwicklungsumgebungen, wie beispielsweise eine Formularbasierte Entwicklungsumgebung oder BASIC-ähnliche Werkzeuge. Man kann sogar direkt auf dem PalmOS kleinere Programme schreiben — eine Spielerei.

1.3 Symbian EPOC (Psion, Nokia, Ericsson, Motorola, Matsushita)

Das 1980 gegründete englische Unternehmen *Psion*[psion] begann 1982 mit der Herstellung von Handheld-Computern und wurde innerhalb von drei Jahren in diesem Bereich Marktführer. 1998 wurde die Softwareentwicklung der Firma Psion ausgegliedert und zusammen mit den Firmen Nokia, Ericsson, Motorola, Matsushita die Firma Symbian Ltd. gegründet, die nunmehr für die Weiterentwicklung und Pflege des Betriebsystems EPOC verantwortlich ist.

1.3.1 Betriebssystemcharakteristika

- Tastatureingabe oder Touchscreen (Stift)
- 32Bit, Multitasking
- Konvertierung zwischen PC und Psiondaten vom Betriebsystem unterstützt

[II] Stand 8/2001

Es gibt unterschiedliche EPOC-Plattformen, die die Hersteller an ihre Produkte anpassen. Der Kern bleibt der gleiche, aber die sogenannten „Profile“ oder „Dialekte” sind unterschiedlich. Es existieren folgende „Dialekte“:

- *Pearl* ist die EPOC-Version für Smartphones
- *Crystal* ist die EPOC-Version für VGA-Anzeige mit Tastaturunterstützung und dient als Referenzdesign für den Nokia Communicator
- *Quartz* ist EPOC-Version für VGA-Anzeigen mit Stift

1.3.2 GUI

Abbildung 3: Symbian Quartz - GUI - Quelle: Infosync

Das GUI ist je nach EPOC-Version (Pearl, Crystal und Quarz) auf Tastatur oder Eingabestift ausgelegt. Es hat eine Menüleiste, die bei Bedarf eingeblendet wird. Icons werden für Programmaufrufe verwendet, jedoch liegt der Schwerpunkt auf der Tastatursteuerung. Es werden spezielle Tasten für Programm- und Menüaufrufe unterstützt.

1.3.3 API

Das EPOC-API bietet Funktionen für C++ Entwickler, Java Entwickler und OPL (Organizer Programming Language) Entwickler.

1.3.4 Kompatibilität

Symbian besitzt ein eigenes internes Datenformat, welches für folgende Anwendungen konvertiert werden kann: Microsoft Outlook, Microsoft Excel, Lotus Organizer, CSV- und TXT-Format. Diese Dateien können beliebig konvertiert werden und stehen direkt in den Anwendungen auf dem Symbian-Gerät zur Verfügung. Seitens der Entwicklung von Software und Abgleich der Daten mit einem PC stehen APIs für Visual Basic, C/C++ und Delphi zur Verfügung.

Auf der EPOC-Plattform sind folgende Applikationen verfügbar:[psion]

- Word, Tabellenkalkulation, Rechtschreibprüfung

- Kontaktmanager, Kalender, Weltkarte und Wecker
- E-Mail, Fax, SMS, Webbrowser, serielle Kommunikationsschnittstellen
- Rechner, Klangrekorder
- Im Windows-Explorer wird das Filesystem des EPOC-Gerätes integriert und kann per „drag & drop“ angesteuert werden.

Applikationen können mit C++, Java und OPL entwickelt werden. Für diese Sprachen stehen Entwicklungsumgebungen zur Verfügung.

1.3.5 Hardware

Diese Geräte stehen momentan für das EPOC-Betriebssystem zur Verfügung: Psions Serie 5mx, netBook, Ericsson MC218, Psion Serie7 (netBook), Psion Revo und als erstes Mobiltelefon der Nokia Communicator 9210 mit Crystal. Anders als beim PalmOS, aber ähnlich Windows CE, sind die Anforderungen an die Hardware nicht speziell umrissen.

1.3.6 Erweiterbarkeit

Die Profile der EPOC-Betriebssysteme sind auf die Geräte angepasst. Die meisten EPOC-Geräte lassen sich entweder per Compact Flash Speicherkarte, PCMCIA, IBM MicroDrive oder per MultiMediaCard erweitern und über serielle Schnittstellen oder Infrarot mit dem Desktop-PC oder Laptop verbinden.

1.3.7 Entwicklungswerkzeuge

Das EPOC-Betriebsystem Version 5 enthält eine JavaVM. Somit können Anwendungen in Java entwickelt werden. EPOC unterstützt Java 1.1.4 bis 1.1.8. Darauf und auch auf die Entwicklungswerkzeuge dazu wird in den Kapiteln über MExE näher eingegangen.

Seit dem Psion Organizer II können Anwendungen mit OPL entwickelt werden.

1.4 Verbreitungsgrad

Der Verbreitungsgrad von Windows CE ist in Europa tendenziell steigend und kommt fast an den des PalmOS heran. Letzteres hatte lange die Marktführerschaft inne. Symbian mit EPOC spielt inzwischen eine untergeordnete Rolle, dürfte aber mit der Markteinführung des Nokia Communicator 9210 in Zukunft wieder aufholen. Der Absatz von EPOC-Geräten sank laut Dataquest[dataq] von 86.000 auf 49.925. Diese Einbussen rühren angeblich vom PSION-Markt her. Der Trend in Richtung WindowsCE ist hauptsächlich auf den erhöhten Absatz im zweiten Quartal 2001 zurückzuführen. Compaq verkaufte laut Dataquest[dataq] in Europa 153.112 Geräte und verachtfachte damit seinen Marktanteil gegenüber dem Vorjahr auf 30,2 Prozent. Durch den starken Anstieg bei Compaq besitzen PalmOS und WindowsCE laut Dataquest[dataq] nun fast identische Marktanteile.

2 Technologien zur Applikationsentwicklung

Es gibt derzeit eine Reihe von realisierten Service Plattformen, um Mehrwertdienste für Mobiltelefone anzubieten. Das folgende Kapitel beschreibt eine der standardisierten Plattformen: MExE[mexef]

2.1 Mobile Execution Environment (MExE)

Nachfolgend werden die relevanten Aspekte des durch das 3rd Generation Partnership Project[3gpp] (3GPP) spezifizierten MExE präsentiert. Aktuelle Informationen können auf der MExE Web Site[mexef] eingesehen werden. Ein Stolperstein bei MExE könnte der Fokus auf Java sein, da Microsoft Java nicht mehr direkt unterstützt.

2.1.1 Was ist MExE?

Da abzusehen ist, dass für die durch mobile Endgeräte zu erwartende Informationsvielfalt unterschiedliche und auf die jeweiligen Bedürfnisse angepasste Geräte produziert werden. MExE soll den Spagat schaffen, einen gemeinsamen Standard zu definieren, damit die gebotenen Informationen und Anwendungen auf unterschiedlichen Endgeräten verwendet werden können.

Der Wunsch nach einem einheitlichen Standard für die Darstellung von Informationen und die Ausführung von Anwendungen auf mobilen Endgeräten, ist jedoch unrealistisch. Schon jetzt gibt es zahlreiche Standards wie beispielsweise WAP und HTTP, die auf mobilen Endgeräten verfügbar, jedoch nicht untereinander kompatibel sind. In der MExE–Spezifikation werden drei unterschiedliche Classmarks (Klassifikationen) vorgeschlagen und beschrieben.

In jedem Classmarks wird festgelegt, wie die Übermittlung der Fähigkeiten eines Endgerätes stattfindet und wie der Server darauf reagiert. Im MExE-Kontext wird dazu der Begriff „capability negotiation“ eingeführt. Es dient dem Zweck, Informationen und Applikationen individuell auf die Bedürfnisse des anfragenden mobilen Endgerätes anzupassen. Dies geschieht entweder dadurch, dass für jedes Classmark eine eigene Version der Information oder Applikation vorliegt, oder diese dynamisch je nach Anfrage und Eigenart des anfragenden Endgerätes aus den Informationen generiert. Für diese „capability negotiation“ wird das http–Protokoll[http] (Classmark 2 / 3) bzw. WAP–Protokoll[wap] (Classmark 1) verwendet.

Weiterhin erfolgt eine sogenannte *“content negotiation”*, die sich auf das zu übertragende Format des Inhalts bezieht. So kann serverseitig ein für das Endgerät optimiertes Format gesendet werden. Beispielsweise bekommt ein Classmark 1-Endgerät (WAP) ein WML–Dokument geliefert, während ein Classmark 2-Endgerät ein HTML–Dokument bekommt.

2.1.2 Die MExE Kategorien (Classmarks)

Wie oben skizziert ist das Ziel von MExE, Entwicklern, Netzbetreibern und Herstellern Konventionen und Standards vorzugeben, mit denen sie Applikationen entwickeln und Inhalte auf diversen mobilen Endgeräten darstellen können.

Die für die einzelnen Kategorien entwickelten Anwendungen sollen auf allen MExE-tauglichen Geräten dieser jeweiligen Kategorie zu verwenden sein und die Informationen auf allen MExE–kompatiblen Geräten dieser Kategorie lesbar sein.

2.1.3 Classmark 1

Classmark 1, basiert auf WAP[wap] (Wireless Application Protokoll). Daher kommt es mit begrenzten Ein- und Ausgaberessourcen wie einer 4 x 18 Pixel großen Anzeige und einer 12 Ziffern Tastatur aus.

WAP ist als weltweiter Standard eingeführt der es ermöglichen soll, auf der Empfängerseite mit geringen Hardwareanforderungen sowie geringen Bandbreiten auszukommen und dennoch Daten in akzeptabler Form zu präsentieren. So muss beispielsweise WML (Wireless Markup Language, die Sprache, in der WAP–Anwendungen geschrieben sind) strengeren syntaktischen Anforderungen als beispielsweise html genügen, um keine unnötigen Daten transportieren zu müssen. Bevor WML über das drahtlose Netz übertragen und auf das mobile Endgerät geschickt wird, wird WML komprimiert um die Datenmenge weiter zu reduzieren.

Classmark 1, welches im wesentlichen WAP ist, hat nach unserem oben eingeführten Abstrakten Model folgende Layer - Charakteristika:

User Input Layer (UIL). WML-Forms um Informationen ein zu geben. Diese bestehen im Wesentlichen aus Texteingabefeldern, Auswahlfeldern und Buttons zum Bestätigen der Eingaben. Links können nach dem Domainnamen und der aufzurufenden Datei mit Informationen angereichert werden, die übertragen werden. z.B.: http://wap.mydomain.com/index.wml?myName=Mustermann

Presentation Layer (PL). WML, WMLScript. Informationen werden bei WAP mittels WML (Wireless Markup Language) komprimiert übertragen und diese von dem mobilen Endgerät dargestellt. WML ist html sehr ähnlich. WML gehört zu der SGML – Familie, die mit sog. Tags arbeitet. Tags sind Meta Informationen, die in spitzen Klammern stehen. Diese wiederum rahmen die eigentlichen Daten ein oder stehen als alleinige Tags für eine bestimmte Anweisung wie beispielsweise ein Zeilenumbruch. Diese Tags sind bei WML sowie HTML für die Auszeichnung der Daten zuständig, die dann von der Software (WAP-Browser) der Endgeräte dargestellt werden.

WMLScript ist eine Scriptsprache, die eingebettet in WML–Seiten verwendet wird. Sie ist teil der WAP–Spezifikation. WMLScript ist eine „light"-Version von JavaScript. WMLScripts sind nicht in WML – Seiten eingebettet. WML–Seiten verweisen lediglich auf URLs die WMLScript enthalten. WMLScript wird in bytecode übersetzt, bevor es an den WAP–Browser geschickt wird. Mit WMLScript wird, ähnlich wie mit JavaScript ein Teil der Logik auf den Client übertragen und dort ausgeführt. Eine Anwendung hierfür ist Beispielsweise Benutzereingaben überprüfen und bestätigen.

Business Logic Layer (BLL). Der BLL ist die Hardwarespezifische Umsetzung des WAP–Browsers und dessen Interpretation von WML bzw. WMLScript. Hier findet die Umsetzung der Logik statt.

2.1.4 Classmark 2

Classmark 2 basiert auf dem Personal Java Environment von Sun. Classmark 2 benötigt eine Hardwareumgebung, die wesentlich mehr Ressourcen, wie Rechenleistung, Speicherplatz und Netzwerkbandbreite benötigt sowie höhere Anforderungen an die grafische Darstellung stellt, dafür mächtigere und flexiblere Anwendungen zulässt. Die Classmark 2 Definition beinhaltet Personal Java mit dem JavaPhone API.

Die Personal Java Umgebung ist eine Java Umgebung, die für mobile Endgeräte angepasst bzw. optimiert wurde, so dass auf diesen Web-Inhalte und Java Applets darstell- oder ausführbar sind. Personal Java basiert auf J2SE (Standard Java 2) mit einigen Standard-Java-Klassen und einigen auf die Bedürfnisse mobiler Endgeräte angepassten Java-Klassen, wie beispielsweise eine modifizierte Version des AWT[awt], welche auf kleinen Geräte ohne Mausunterstützung und mit wesentlich kleineren Anzeigen als ein PC auskommen muss.

Die JavaPhone API ist eine Java-Erweiterung, die insbesondere folgende Funktionen anbietet: Zugriff auf die Telefonfunktionen, die Adressbuchfunktion, SMS–Nachrichten, eine Kalenderfunktion, Benutzerprofile, Batteriestatusüberwachung und das Installieren von Anwendungen. Classmark 2 (sowie auch Classmark 3) benötigen das HTTP-Protokoll, um *capability*- und *content negotiation* durchzuführen. Auch das Herunterladen von Dateien und Anwendungen erfolgt bei MExE über das HTTP-Protokoll.

User Input Layer (UIL). Der UIL besteht aus den Eingabekomponenten des AWT wie Checkbox, Button, TextField und TextArea. Damit können einfache GUIs erstellt werden, die mit Tastatur, einer Mausemulation oder einem Eingabestift bedient werden können. Die Texteingabefelder können mehrzeiligen Text aufnehmen.

Presentation Layer (PL). Der PL besteht aus Komponenten des AWT wie TextField, TextArea, Label und Scrollbar. Diese Elemente unterstützen (wie oben) einfache GUIs auf der Ausgabeebene. Es ist möglich, farbige Grafiken und Tabellen in begrenztem Umfang darzustellen.

Business Logic Layer (BLL). Der BLL hat die Mächtigkeit des PJAE zusammen mit dem JavaPhone API.

PJAE - PersonalJava Environment

Das PJAE[pjae] ist eine Java Umgebung, die in Java geschriebene Software ausführen kann. PJAE ist für Software auf netzwerkfähigen mobilen Endgeräten zugeschnitten. PJava basiert auf dem JDK 1.1.6 und wurde um folgende Aspekte erweitert:

- Double Buffering – Optimierung des Updates bei grafischen Operationen, die die Anzeige verändern
- Eingabemöglichkeiten ohne Maus: dazu gehören Schnittstellen wie NoInput-Preferred, KeyboardInputPreferred, ActionInputPreferred, PositionalInputPreferred. Sie beschreiben, wie Benutzer, die keine Maus besitzen, mit dem System interagieren können.
- Timer API – Klassen, die Ereignisse zeitgesteuert auslösen.

Die JavaPhone-API stellt alle für ein Mobiltelefon wichtigen Telephonie-Funktionen zur Verfügung. Hierfür gibt es in Java keine Alternative.

2.1.5 Classmark 3

Classmark 3 - Geräte basieren auf J2ME[j2me] CLDC[cldc] mit dem MIDP[midp]. J2ME ist eine Version von Java, die auf mobile Endgeräte angepasst wurde. CLDC besteht aus einer JavaVM (in diesem Falle die KVM) und APIs, die dafür bestimmt sind, eine Umgebung zu erstellen, die mit limitierten Hardwareanforderungen, wie etwa geringer Rechenleistung, geringem Speicherplatz und geringer Bandbreite, auskommt.

User Input Layer. Der UIL besteht aus Eingabekomponenten des MIDP wie ChoiceGroup und TextField. Diese sind wenige und eingeschränkte Klassen, die nicht die Mächtigkeit derer von Classmark 2 haben. Mit ihnen lassen sich einzeilige Texte (TextField) eingeben und aus einem Menü auswählen (ChoiceGroup).

Presentation Layer. Der PL besteht auch aus Java-Komponenten des MIDP wie StringItem, Ticker, ImageItem und Gauge. Auch diese sind auf die beschränkten Hardwareressourcen abgestimmt. StringItem kann einen einfachen Text darstellen. Ticker kann einen einfachen Text als Laufschrift darstellen. ImageItem stellt einfache s/w-Grafiken dar und Gauge biete die Darstellung einfacher Balkendiagramme.

Business Logic Layer. Der BLL hat die Mächtigkeit der J2ME Plattform, welche als wesentlichen Bestandteil die KVM enthält. Die KVM ist eine JavaVM, die speziell auf die Bedürfnisse sehr eingeschränkter mobiler Endgeräte zugeschnitten ist. Diese sind geringer Speicherbedarf, geringer Energiebedarf und geringe Rechenleistung.

Die Mindestanforderungen an die Hardware für eine MIDP-Umgebung (und damit für Classmark 3) sind folgende

Display

- Bildschirmgröße 96 x 54 Pixel
- Anzeigenfarbtiefe 1 Bit
- Pixeldimensionen 1x1

Tastatur

- Einhand-Eingabefeld oder
- QWERTY(Z) - Tastatur oder
- Touch Screen

Speicher

- 128 KB beständiger Speicher für MIDP Komponenten
- 8 KB beständiger Speicher für Daten, die von den Anwendungen erzeugt und verwendet werden
- 32 KB Speicher für die Java Laufzeitumgebung

Netzanbindung

- Vollduplex

- Drahtlos
- Begrenzte Bandbreite (9600 Bps)

2.1.6 Unterschiede zwischen den Classmarks

Classmark 1 - Classmark 2 / 3

Der wesentliche Unterschied zwischen Classmark 1 und den Classmarks 2 / 3 ist das zugrundeliegende Protokoll. Classmark 1 verwendet WAP, Classmark 2 und 3 verwenden HTTP zur Übertragung der Daten.

Classmark 2 / Classmark 3

Der Hauptunterschied zwischen Classmark 2 und Classmark 3 besteht im Umfang der Java Version. Classmark 2 basiert auf dem JDK 1.1.6 und den JavaPhone-Klassen. CLDC (Classmark 3) benutzt die KVM, also eine wesentlich eingeschränktere JavaVM. Intuitiv anzunehmen wäre, dass die Classmarks eine Hierarchie bilden. Dem ist allerdings **nicht** so. Bei Classmark 2 sind die Hardwareanforderungen wesentlich höher als bei J2ME CDLC (Classmark 3). Classmark 2 benötigt beispielsweise eine Anzeige, die das AWT darstellen kann. Allein diese Bedingung verlangt nach leistungsfähigerer Hardware als die meisten der bisher bekannten Mobiltelefone und Classmark 3-Geräte sie bieten. Eines der Geräte, das Classmark 2 unterstützt, ist beispielsweise der Nokia Communicator 9210.

2.1.7 Tauglichkeit der Classmarks

Classmark 1

Classmark 1 eignet sich am besten für Mobiltelefone ohne größeres Display. Classmark 1 ist WAP, welches die meisten Hersteller bereits in ihre Geräte integriert haben.

Classmark 2

Classmark 2 eignet sich am besten für Smartphones und PDAs mit Kommunikationsanbindung. Durch die höheren Anforderungen an die Hardware und das Display ist es kaum möglich und wenig sinnvoll, sie in einem herkömmlichen Mobiltelefon unterzubringen.

Classmark 3

Classmark 3 eignet sich am besten für Mobiltelefone mit begrenzten Ressourcen. Da die Anforderungen an die Hardware gering gehalten sind, ist es gut möglich, kleinste Geräte basierend auf Classmark 3 zu bauen. Im Unterschied zu Classmark 1 ist es bei Classmark 3 möglich, Anwendungen zu entwickeln, die mehr Logik auf dem Telefon ausführen als mit es reinem WAP (Classmark 1) möglich währe.

2.2 Fazit und Ausblick

Wie eingangs erwähnt, ist ein Problem von MExE die Konzentration auf Java. PDAs mit Windows-Betriebssystem werden auf .NET und der Programmiersprache C# aufbauen. Generell ist zu hinterfragen, ob die Stoßrichtung, Anwendungsentwicklung auf mobilen

Geräten zu standardisieren, überhaupt sinnvoll ist. Das zur Verfügung stellen eines Web-Browsers reicht für Web-basierte Anwendungen.

Es wird abzuwarten sein, wie sich diverse mobile Endgeräte am Markt durchsetzen, die MExE–kompatibel sind. Bedarf nach leistungsfähigeren Geräten besteht und so gut wie alle größeren Hersteller haben Geräte, die Classmark 2 oder Classmark 3 unterstützen, entweder angekündigt oder bereits auf dem Markt. Da selbst WAP-fähige Geräte beträchtliche Unterschiede aufweisen, zum Beispiel was die anzeigbare Dokumentlänge betrifft, sind auch bei Classmark 2 und 3 verschiedene Implementierungen zu erwarten. Eine hinreichend exakte Standardisierung der Kategorien wird voraussichtlich nicht erfolgen. Selbst bestehende Spezifikationen, wie die *Capability Negotiation* sind nicht umgesetzt.

3 Anwendungsentwicklung - Werkzeuge und Praxistauglichkeit

Dieses Kapitel zeigt auf, welche Werkzeuge für die Anwendungsentwicklung unter MExE eingesetzt werden.

3.1 Anwendungsentwicklung mit MExE

In diesem Abschnitt werden Werkzeuge zur Entwicklung von MExE-Anwendungen vorgestellt. Außer der Verfügbarkeit und den Kosten werden die jeweiligen Besonderheiten skizziert.

3.1.1 Suns Referenzimplementation J2ME CLDC

Unter[cldcr] stellt Sun die Referenzimplementation für J2ME CLDC zur Verfügung. Diese beinhaltet Binärdateien sowie den Sourcecode, um Entwicklern und Herstellern die Möglichkeit zu geben, die CLDC Implementierung auf neue Geräte zu portieren und um Anwendungen zu entwickeln.

Verfügbarkeit und Kosten. Die Entwicklungsumgebung wird von SUN[sun] kostenlos unter der Community Source Lizenz für Entwickler bereitgestellt. Sobald das damit entwickelte Produkt kommerziell genutzt wird, müssen entweder für die Technologie Lizenzgebühren bezahlt werden und / oder für das Logo Warenzeichengebühren entrichtet werden.

Besonderheiten. Die Referenzimplementation wird von SUN im Rahmen der Community Source Lizenz elektronisch zum Herunterladen bereitgestellt. SUN ist bemüht, die Java Plattform so weit wie möglich für Anwender und Entwickler zu verbreiten. Von SUN wird deshalb viel Wert auf umfangreichen Support gelegt. SUN geht Kooperationen mit Herstellern ein und bindet diese in die Entwicklung neuer Technologien ein.

Programmiersprachen. J2ME basiert auf Java.

3.1.2 Borland JBuilder / Nokia Mobile Set

Der JBuilder mit integriertem Nokia Mobile Set wird von Nokia[nokia] und Borland[borl] kostenlos auf deren Internetseiten bereitgestellt. Es beinhaltet die Java Entwicklungsumgebung JBuilder[jbuil] und darin integriert Werkzeuge für die Erestellung von Classmark 2 kompatiblen Applikationen.

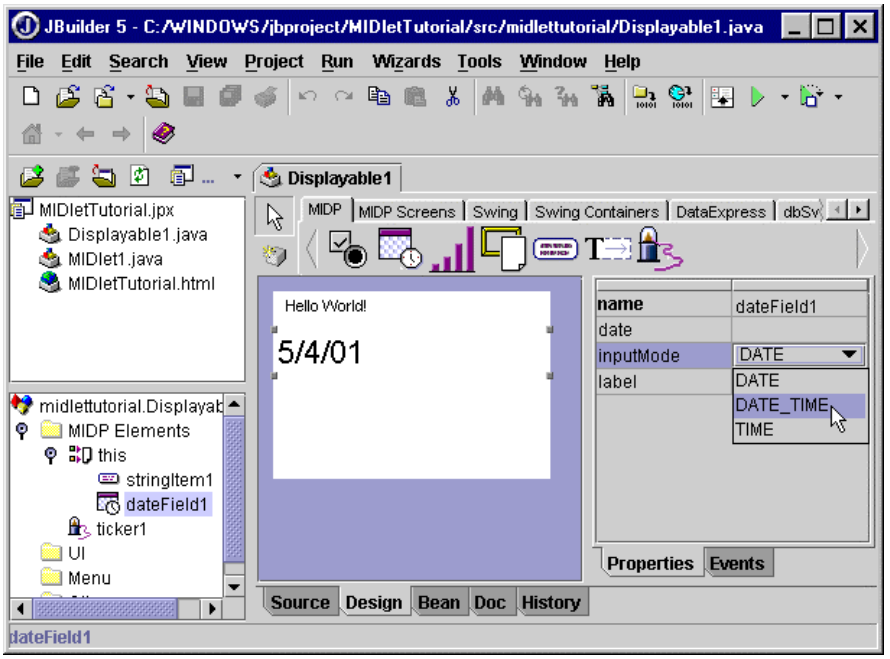

Abbildung 4: Nokia Mobile Set

Verfügbarkeit und Kosten. Der JBuilder von *Borland* wird in der Personal Edition in Verbindung mit dem Nokia Mobile Set frei zur Verfügung gestellt, sofern keine kommerzielle Entwicklung erfolgt. Vorraussetzung ist eine Registrierung bei *Borland*. Der JBuilder5 und das Mobile Set können von der Internetseite von *Borland* bezogen werden.

Besonderheiten. Der JBuilder liegt in der Version 5[III] vor und ist ein verbreitetes Tool für Java Entwicklung. Die MIDP Erweiterungen sind als grafische Tools in diese Umgebung integriert.

3.1.3 Nokia Symbian Crystal SDK for Java

Das Crystal SDK (Software Development Kit) für die Symbian Plattform wird von Nokia speziell für den Nokia Communicator 9210 kostenlos bereitgestellt. Es beinhaltet einen Emulator für den Nokia Communicator 9210, auf dem die erstellten Programme getestet werden können. Des weiteren enthält es Entwicklungswerkzeuge und eine ausführliche Anleitung zur Softwareentwickelung. Die so erstellte Anwendung kann über das Internet oder auf Datenträgern verteilt werden und mit der im Lieferumfang enthaltenen Software des Nokia Communicator 9210 auf diesem installiert werden.

[III] Stand 08/2001

Abbildung 5: Crystal SDK Emulator (Screenshot)

Verfügbarkeit und Kosten. Das SDK ist von Nokia kostenlos erhältlich. Vorraussetzung ist eine Mitgliedschaft im Forum Nokia[fnok]. Jede beliebige Java-Entwicklungsumgebung kann verwendet werden, wie z.B. der *JBuilder*[jbuil] von *Borland*, der wie erwähnt in der Personal Version kostenlos erhältlich ist. Die Tools, die dem Entwickler zur Verfügung gestellt werden, sind auf einer CD enthalten und umfassen alles Notwendige, um Anwendungen zu entwickeln und zu verteilen. Nokia ist bemüht, eine Entwickler-Gemeinschaft aufzubauen. So werden Wettbewerbe ausgeschrieben, wie der *German Nokia 9210 Communicator Application Contest.*

Besonderheiten. Anwendungen werden in Java entwickelt. Die zusätzlichen Symbian-Klassen, um die EIKON-Buttons (seitliche vier Buttons des Communicators) anzusteuern, sind wie die Event-Klassen des AWT zu benutzen. Leider bestehen keine integrierten, auf das SDK abgestimmten Erweiterungen für den JBuilder für diese.

3.2 Marktüberblick über MExE Endgeräte

Es gibt, mit Ausnahme von Classmark 1 (= WAP), auf dem Markt noch wenige Smartphones, die den MExE Standard erfüllen. Angekündigt haben so gut wie alle Mobiltelefonhersteller Geräte, die entweder zu Classmark 2 oder zu Classmark 3 kompatibel sein sollen.

3.2.1 Unterstützung von Classmark 1

Prinzipiell unterstützen alle Mobiltelefone, die WAP unterstützen, auch Classmark 1 (d.h. WAP = Classmark 1). Die Spezifikation von Classmark 1 beinhaltet Push-Technologie, die vom WapForum in der Version WAP 2.0 vorgegeben wurde. In der Praxis wird diese Technologie jedoch kaum umgesetzt.

3.2.2 Unterstützung von Classmark 2: Nokia Communicator 9210

Abbildung 6: Nokia Communicator 9210

Der Nokia Communicator 9210 unterstützt PersonalJava sowie das HTTP–Protokoll für Capability Negotiation und erfüllt damit die wichtigsten Voraussetzungen für Classmark 2.

Mit dem Nokia Communicator 9210 lassen sich eine Vielzahl mobiler Anwendungen nutzen. Der Nokia Communicator 9210 bildet eine mobile Kommunikationsplattform. Er bietet Internet-Zugang per integriertem Web–Browser (zusätzlich zu WAP), Kommunikation per E-Mail (pop3), SMS und Fax (G3), Office-Anwendungen, sowie Kalender- und Kontaktverwaltung. Zusätzlich lässt sich ein RealPlayer installieren, jedoch kein Flash- oder sonstige Plugins für den Browser. Der Nokia Communicator 9210 verfügt über ein hochauflösendes Display mit 4096 Farben, mit dem Web-Seiten aus dem Internet mit Frames und eingebundenen Java–Applets in guter Qualität und, dank HSCSD, in akzeptabler Zeit, betrachtet werden können. Es können Word-Dokumente und Excel-Tabellen editiert und versandt werden. E-Mail-Attachments können empfangen werden und Dateien aus dem Internet geladen werden. Bilder können von einer Digitalkamera per Infrarot in den Nokia Communicator 9210 übertragen und von dort aus angezeigt und per Fax oder E-Mail versendet werden.

Der Nokia Communicator 9210 ist per MultiMediaCard erweiterbar, um zusätzliche Software einzusetzen. Auf dieser MultiMediaCard ist auch die JavaVM untergebracht.

Für den Nokia Communicator ist eine MIDP–Simulation verfügbar, die den Communicator auch Classmark 3 kompatibel macht. Diese liegt jedoch erst in der Beta-Version vor.

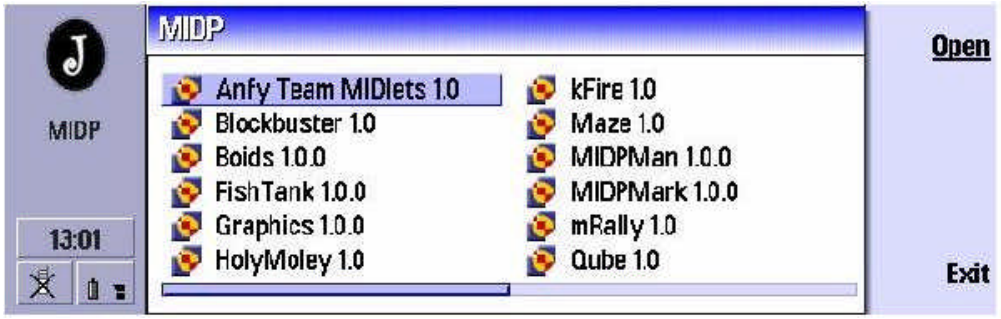

Abbildung 7: MIDP-Umgebungsemulation auf Nokia Communicator 9210

Fazit

Geräte wie der Nokia Communicator, die Classmark 2 unterstützen, haben von ihrer Handhabung und Anforderungen an die Hardware nur noch wenig mit einem herkömmlichen Mobiltelefon gemeinsam. Sie sind in die Kategorie „Mobiler Computer/Personal Digital Assistant mit Telefonfunktion" einzuordnen. Auffallend ist, dass die mit dem Communicator mitgelieferten Anwendungen nicht mit Java/MExE, sondern auf Basis des EPOC-APIs erstellt wurden.

3.2.3 Classmark 3 Geräte

Bisher sind folgende Classmark 3-Geräte angekündigt beziehungsweise am Markt verfügbar:

MotorolaAccompli008

Das MotorolaAccompli008[motorola] unterstützt J2ME und damit Classmark 3. Laut Herstellerangaben unterstützt das MotorolaAccompli008 J2ME und WAP, jedoch ist kein Web–Browser integriert, was vermuten lässt, dass analog zum Siemens SL45i die Anwendungen per WAP oder alternativ per PC auf das Mobiltelefon zu übertragen sind. Des weiteren gibt der Hersteller an, Sprach-, Daten- und Internet-Tools kombinieren zu können.

Siemens SL45i

Das Siemens SL45i[siemens] unterstützt den Wireless Java[wjava] – Standard. Siemens war an der Entwicklung als Partner von Sun beteiligt. Das Siemens SL45i unterstützt MIDP. Laut Angaben von Siemens[siemens] soll der Netzwerkbetreiber zum Zeitpunkt des Verkaufes ein Anwendungspaket anbieten, das vor Ort auf dem Mobiltelefon installiert wird.

Danach können Anwendungen, wie beispielsweise Spiele, ortsabhängige Dienste sowie Verzeichnisse auf das Gerät übertragen werden. Dies kann auf zwei verschiedenen Wegen geschehen:

- Es wird eine WAP–Seite, die eine MIDP-konforme Java Anwendung anbietet, mittels des Mobiltelefons besucht und die Anwendung von dort aus auf das Mobiltelefon übertragen. Dies geschieht über das Mobilfunknetz.

- Die MIDP–konforme Anwendung wird über das Internet auf den PC und von dort aus mittels seriellem Kabel auf das Mobiltelefon übertragen.

Das Mobiltelefon enthält eine Möglichkeit zur Speichererweiterung mittels einer austauschbaren MultiMediaCard (MMC). Diese ist derzeit mit 128 MB erhältlich[IV]. Sie kann beliebig ausgetauscht werden. Auf ihr werden auch MP3–Dateien für den im Mobiltelefon integrierten MP3–Spieler gespeichert.

Im Sinne einer Definition der Classmarks fällt das Siemens SL45i in die Kategorie Classmark 3.

Fazit

Classmark 3 - Geräte müssen wegen der Hardwarebeschränkungen (MIDP) mit wesentlich weniger Ressourcen auskommen. Im Mittelpunkt steht hier weiterhin die Mobiltelefonie, die jedoch um die Classmark 3-Funktionalität erweitert ist. Es wird bezweifelt, ob mit Classmark 3 ein Durchbruch bei datenbasierten Diensten erzielt werden kann. Dadurch, dass es mit einfachen und frei verfügbaren standardisieren Mitteln (Java) leicht möglich ist, kleinere Anwendungen zu entwickeln und diese zu verteilen, könnte ähnlich wie beim PalmOS ein Markt für kleine Anwendungen und Tools entstehen.

[IV] Stand 4/2002

4 Softwareverteilung

Mit Softwareverteilung sind sowohl die Installation einer Applikation sowie die darauf folgenden Updates gemeint.

4.1 Softwareverteilung für MExE

In diesem Abschnitt wird dargelegt, wie Software und Dienste auf MExE-Endgeräten verteilt und installiert werden.

4.1.1 MExE Service Environment (MSE)

Das MExE Service Environment besteht aus MExE Endgeräten, die über ein Netzwerk miteinander verbunden sind. Das Netzwerk besteht aus drahtlosen (z.B. Bluetooth), Festnetz- und mobilen Verbindungen, die beliebig miteinander kombiniert werden können. Der Netzbetreiber stellt das Netzwerk für sprachbasierte-, Multimedia-, Daten- und sonstige Dienste bereit. Die MExE-Anbieter stellen Dienste zur Verfügung, die auf dem MExE-Standard basieren. Die Dienstanbieter und Endgerätehersteller sind dafür verantwortlich, die MExE-Standards einzuhalten. Auf den Netzbetreiber kommen damit außer der Bereitstellung der Infrastruktur keine weiteren Aufgaben zu, die MExE betreffen.

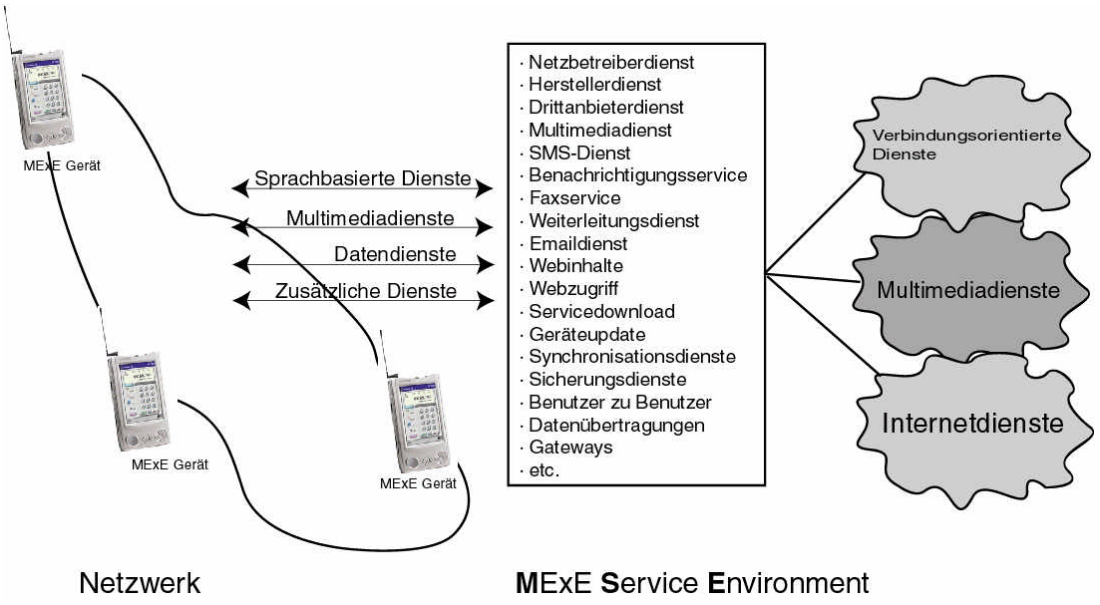

Abbildung 1. MExE Service Environment nach der 3GGP-Functional Description
Quelle: [3gpp]

4.1.2 MExE-Server

Als MExE-Server kann jeder beliebige Webserver im Internet dienen. Es werden WML, WMLScript und Java Programme zur Verfügung gestellt. Das MExE-Endgerät verbindet sich per HTTP oder WAP mit dem Server, um so die Daten oder Programme abzurufen. In einem solchen Kontext besteht keine Notwendigkeit für den Netzwerkbetreiber, spezielle MExE-Infrastruktur bereitzustellen.

MExE Dienste können auf mehreren MExE-Servern verteilt sein. Diese können untereinander verbunden sein. Die Knotenpunkte, die MExE–Services anbieten, benutzen bestehende Übertragungsarten. Die sind beispielsweise Übertragungsprotokolle für Festnetzverbindungen, drahtlose Verbindungen, wie beispielsweise Infrarot oder Bluetooth, Mobilnetzprotokolle wie GSM und UMTS oder Internetprotokolle. Kommen in der Zukunft neue hinzu, können diese genutzt werden, solange sie Daten von A nach B (bzw. auf ein MExE-Endgerät) transportieren.

Die MExE-Server, die MExE-Dienste anbieten, können in beliebigen Umgebungen, wie IP-basierten Umgebungen, verbindungsbasierten Umgebungen, Intranets sowie Internetumgebung integriert sein. Es muss jedoch ein Zugriff auf die Dienste über eine der oben beschriebenen Verbindungen möglich sein. Weiterhin kann zusätzlich ein Wandler eingesetzt werden, der internetspezifische Protokolle in für das Mobilnetz optimierte Formen umwandelt, wie es beispielsweise bei einem *WAP–Gateway* der Fall ist. Dort wird HTML in WML umgesetzt und kompiliert, bevor es an ein mobiles Endgerät geschickt wird.

4.1.3 Capability und Content Negotiation

Capability negotiation ist obligatorisch für alle Classmarks[mexef]. Capability Negotiation heißt, dass sich der Server und das Endgerät verständigen, welche Fähigkeiten (*capabilities*) dieses hat. Der Server reagiert dementsprechend darauf. Für diese „*capability negotiation*“ wird das HTTP-Protokoll (Classmark 2/3) bzw. WAP–Protokoll (Classmark 1) verwendet.

Die Unterstützung des Content Negotiation Processes ist optional.

4.1.4 Anforderungen an den Netzbetreiber

Nur für Classmark 1 ist seitens des Netzbetreibers ein WAP-Gateway notwendig. Für alle anderen MExE Dienste wird ausschließlich das HTTP-Protokoll benutzt und so die Einhaltung des Standards den MExE-Anbietern überlassen.

Der Netzbetreiber hat die Aufgabe, eine Infrastruktur für Datendienste zur Verfügung zu stellen, zu pflegen und auszubauen. Diese Aufgabe ist nicht an MExE gebunden. Auch andere Dienste die nicht mit MExE im Zusammenhang stehen, bedienen sich dieser Infrastruktur. MExE-Kompatibilität ist auf Seiten der Endgerätehersteller und der Dienstanbieter gefordert.

4.1.5 Anwendungsverteilung über das Internet

Für Mobiltelefone, die Classmark 2 unterstützen (PersonalJava), ist eine Anwendungsverteilung über das Internet oder andere Datenträger die gängigste Methode.

Dateien werden wie herkömmliche Anwendungen für PCs zum Herunterladen angeboten, auf den PC geladen und zum Beispiel über ein serielles Kabel, Infrarot oder Bluetooth auf das mobile Gerät übertragen.

Diese Art der Übertragung und Verteilung hat den Nachteil, an einen stationären Zugang zum Internet gebunden zu sein. Alternativ dazu können die Applikationen auf einem Datenträger vorliegen, welcher wiederum auf einem PC gelesen und dann in das mobile Endgerät übertragen wird.

Eine weitere Möglichkeit der Verteilung von Software auf mobile Geräte ist das direkte Herunterladen auf das mobile Gerät. Das kann, falls vorhanden, über den integrierten Webbrowser erfolgen. So können beispielsweise Updates direkt über das Mobilnetz geladen werden. Der Nachteil dieser Methode ist der Zeit- und Kostenaufwand, der zumindest derzeit noch hiermit einhergeht.

Der erste Ansatz basiert darauf, dass der Nutzer aktiv nach den Applikationen sucht, um sie zu installieren. Updates werden auf Initiative des Benutzers installiert. Beim zweiten Ansatz kann die Applikation nach dem Start bei der ersten Verbindung mit dem Server prüfen, ob eine neue Version vorliegt und das Update herunterladen.

Java-Anwendungen beispielsweise lassen sich so entwickeln, dass sie selbständig nach einer neuen Version suchen. Dies lässt sich so realisieren, dass sie eine bestimmte Internetadresse in einem vorher festgelegten Format, beispielsweise eine XML-Datei mit Versionsinformationen, abfragt und, falls eine neue Version vorliegt, die Neuerungen anfordert. Jede dieser Anfragen erfolgt mit Einverständnis des Benutzers.

Eine weitere Möglichkeit ist, eine SMS vom Anwendungsanbieter zu versenden, die über eine neue Version der Anwendung informiert. Diese kann so gestaltet sein, dass zum Beispiel eine einfache SMS-Antwort (Ja) genügt, um einen Update über das Mobiltelefon zu erhalten.

4.1.6 Automatische Konfiguration

Heute schon vielfach von Mobilnetzbetreibern verwendet sind sogenannte Konfigurations-SMS. Diese dienen zum Konfigurieren von installierten Anwendungen oder Zugängen. Beispielsweise wird für einen WAP- oder Internetzugang auf einem Mobiltelefon vom Mobilnetzbetreiber eine SMS geschickt, die auf das jeweilige Telefon abgestimmt ist und von diesem lesbar ist. Der Benutzer hat die gesendete Nachricht nur noch zu bestätigen und die Konfigurationsdaten werden an der vorgesehenen Stelle in der Anwendung, beispielsweise dem Internet-Browser, eingefügt.

5 Beispielanwendungen

Im Folgenden wird anhand von einem praktischen Beispiel aufgezeigt, wie eine Anwendung auf unterschiedlichen Plattformen entwickelt wurde. Es werden jeweils Beschreibungen und Screenshots präsentiert. Der vollständige Quelltext ist im Anhang zu finden. Die weiteren Bestandteile, wie das lauffähige Programm und die Installationsdateien für die jeweilige Plattform sind im Internet unter http://bachelorarbeit.gschreiber.com zu finden.

Als Beispielanwendung wurde eine mobile Zeiterfassung mit dem Namen *myTime* implementiert. Folgendes Szenario ist bei dieser Implementierung zu Grunde gelegt worden:

Eine Außendienstmitarbeiter, Handwerker, Selbständiger oder eine beliebige Person, die mobil Zeit erfassen möchte oder muss, würde dies gerne mit einem MExE Endgerät tun. Beispielsweise möchte ein Handwerker die Zeit, die er für einen Auftrag bei einem Kunden benötigt und in Rechnung stellen möchte, mit einem PDA oder seinem Handy erfassen und die so erfasste Zeit zur Weiterverarbeitung an ein System - möglichst mobil – übermittelt werden. Für diese Zeiterfassung auf der Seite der mobilen Geräte wurde nun *myTime* entwickelt und implementiert. Die Schnittstelle zu weiterverarbeitenden Systemen wurde dabei so weit wie möglich aus standardisierten Komponenten aufgebaut, um sie in bestehende Systeme integrieren zu können.

5.1 myTime mit Classmark 1

Zur Entwicklung wurde das Nokia-WAP Toolkit in der Version 3.0 verwendet. Es wurde ausgewählt, weil es kostenlos erhältlich ist und Nokia einen Großenteil der WAP-fähigen Mobiltelefone herstellt.

Das WAP-Toolkit ist nach Registrierung im Forum Nokia[fnok] kostenlos erhältlich.

5.1.1 Konfiguration der Entwicklungsumgebung

Die Entwicklungsumgebungskonfiguration bestand aus folgender Hard- und Software:

PC mit Pentium III, 512MB RAM, CD-Rom, 40GB HD ATA 100, Windows 2000 deutsch mit Sevice Pack 2.

Die Installation unter Windows ist menügesteuert und gestaltet sich sehr einfach. Es wird das Nokia WAP-Toolkit und das JDK 1.3.1 installiert, das vom Nokia WAP-Toolkit als Laufzeitumgebung benötigt wird.

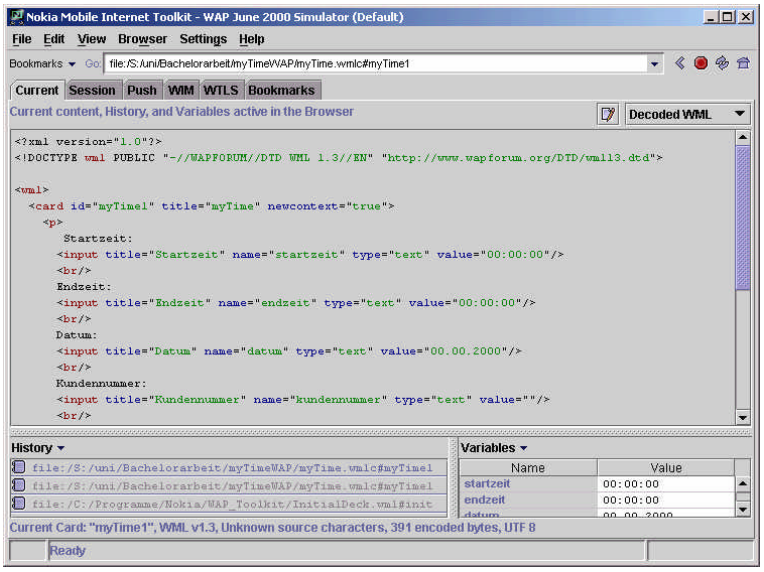

Abbildung 8: Oberfläche des Nokia WAP-Toolkit

Zum Nokia WAP-Toolkit gehört ein Simulator, der die erstellten WAP – Seiten in einem simulierten Mobiltelefon anzeigt.

Abbildung 9: WAP-Simulator

5.1.2 Entwicklung und Struktur von myTime auf Classmark 1

Leider verfügt weder WML noch WMLScript über eine Funktion, die die aktuelle Zeit anzeigen oder speichern kann. Diese kann folglich auch nicht einfach übertragen werden. Es ist somit nicht möglich, mit Classmark 1 *myTime* komplett auf der Clientseite zu realisieren.

Die aktuelle Zeit muss von einem (Web-) Server eingelesen werden oder manuell eingegeben werden. Bei vorliegender Implementierung wird die Zeit manuell eingegeben, um das Beispiel einfach und unabhängig von einem Webserver zu halten.

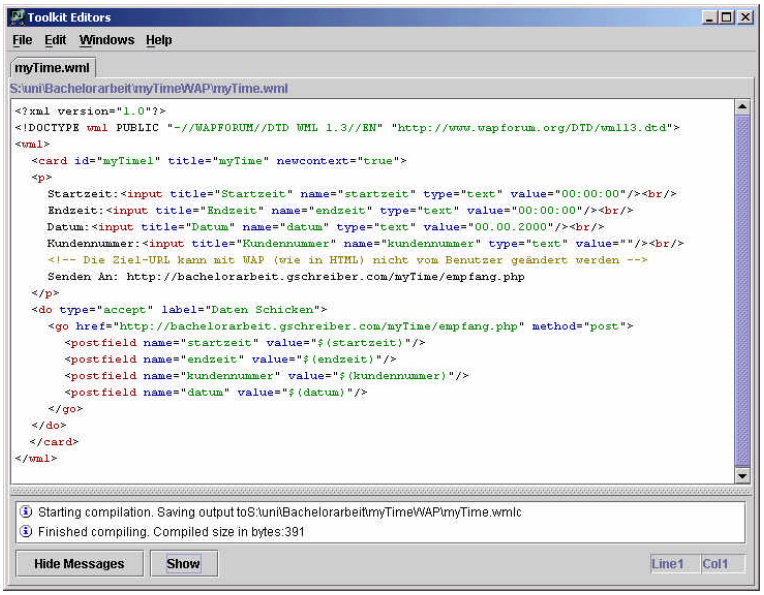

Abbildung 10: Editorfenster im Nokia WAP-Toolkit

Die „Applikation“ besteht aus einem sog. „Deck“, einem Satz von „Cards“ (in unserer Anwendung nur eine). Diese Card enthält die Eingabefelder für die Daten (Startzeit, Endzeit, Datum, Kundennummer) und wird auf der Anzeige zum Ausfüllen angezeigt.

Bei WAP ist es (genau wie bei HTML) nicht möglich den Benutzer über das Formular festlegen zu lassen, an welche Zieladresse die Daten geschickt werden sollen. Deshalb ist eine Eingabe der Zieladresse durch den Benutzer, wie bei den anderen Implementierungen von *myTime*, nicht möglich, sondern wird im Quelltext (und damit auf der Serverseite) festgelegt.

5.1.3 Funktionsweise von myTime auf Classmark 1

Als Eröffnungsseite nach dem Start (Aufruf des Deck, bzw. Abruf der wml „Seite“) erscheint ein Fenster, das Startzeit, Endzeit, Datum, Kundennummer und Zieladresse anzeigt. Bis auf die Zieladresse können die Felder durch Auswahl editiert werden und die gewünschten Werte eingetragen werden.

Abbildung 11: Ausgefüllte Card, bereit zum Absdicken

Sind alle Werte wie gewünscht ausgefüllt, können die Daten an die Zieladresse übermittelt werden.

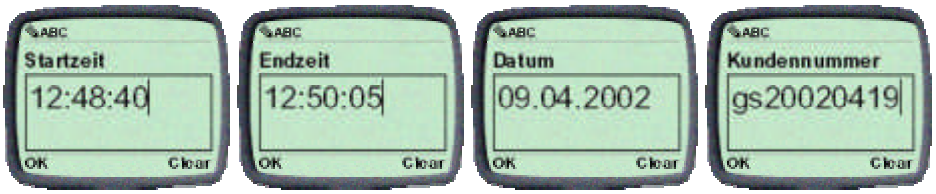

Abbildung 12: Werte manuell eingeben

Abbildung 13: Werte editieren oder alles Abschicken

Der Server sendet eine Antwort, die daraufhin angezeigt wird.

Abbildung 14: Antwort vom Server, welche Daten angekommen sind

5.2 myTime mit Classmark 2

Um *myTime* mit Classmark 2 zu realisieren, wurde der Nokia Communicator 9210 als Zielplattform ausgewählt. Dies deshalb, weil er der Classmark 2 Spezifikation am nächsten kommt, als Testgerät zur Verfügung stand und momentan das einzige auf dem Markt erhältliche Gerät mit PersonalJava Unterstützung ist[mardev]. Er besitzt eine JavaVM und die javaPhone API wird ebenfalls unterstützt. Dies sind die wichtigsten Vorraussetzungen für Classmark 2.

Für die Entwicklung wurde das SDK von Nokia für den Communicator 9210 verwendet. Dieses beinhaltet eine Version für C++ und für Java. Im vorliegenden Fall wurde die Version für Java verwendet.

Das SDK basiert auf Sun's PersonalJava Implementierung und der JavaPhone API. In der Version 0.9 musste das Java SDK ("JavaCompiler") aus lizenzrechtlichen Gründen noch separat von der Webseite von Sun heruntergeladen werden, in der derzeit vorliegenden

Version 1.1[V] ist es bereits auf der Installations-CD enthalten, wird jedoch separat installiert. Wird eine Entwicklungsumgebung wie der JBuilder von Borland verwendet, ist diese Installation nicht mehr nötig. Der JBuilder hat bereits ein SDK integriert. Um jedoch volle Kompatibilität mit der Version des SDK des Mobiltelefons, zu ereichen, ist es ratsam, dieses trotzdem zusätzlich zu installieren.

5.2.1 Konfiguration der Entwicklungsumgebung

Für die Implementierung von *myTime* war die Entwicklungskonfiguration folgende Hard- und Software:

PC mit Pentium III, 512MB RAM, CD-Rom, 40GB HD ATA 100, Windows 2000 deutsch mit Service Pack 2, JBuilder 5 Personal Edition.

Zuerst muss das SDK von Nokia besorgt werden. Dies kann nicht von der Webseite herunter geladen werden, sondern wird auf dem Postweg als CD geliefert. Dazu ist es nötig, sich im *Forum Nokia*[fnok] kostenlos zu registrieren. Nach der Registrierung erhält man Zugriff auf die Supportbereiche für diverse mobile Endgeräte des Herstellers mit Softaredownloads, Foren und unter anderem eben die Möglichkeit das SDK in der momentan vorliegenden Version 1.1[V] zu bestellen. In etwa zwei bis vier Werktagen erhält man dann eine CD, auf der besagtes SDK zu finden ist.

Die Installation ist nur auf einem Windows Betriebsystem möglich. Zunächst wird bei der Installation darauf hingewiesen, dass das die Software nicht auf dem Betriebssystem Windows 2000 getestet wurde und nicht für diese entwickelt wurde. Man hat jedoch trotz dem die Möglichkeit, das SDK auf Windows 2000 zu installieren.

In der Praxis lief das SDK und der Emulator für den Communicator 9210 nach einer Neuinstallation des Betriebssystems fast problemlos (näheres dazu siehe Seite 46, Abschnitt 5.2.6 „Probleme bei der Implemeniterung“).

Die Installation des SDK dauert etwa 20 Minuten, da viele kleine Dateien kopiert werden. Anschließend installiert das Setupprogramm das SDK 1.1.8 von Sun und ActiveStatePerl von einem weitern Drittanbieter.

Bei der gesamten Installation wurden die vorgegeben Installationspfade übernommen.

Das SDK besteht aus folgenden Teilen:

- Java SDK 1.1.8 - Dieses wird benötigt, um die Quelldateien mittels dem Befehl *javac JavaQuelldatei* in von der Java VM ausführbare Klassendateien (.class - Dateien) um zu wandeln
- AIF Icon Designer - wird benötigt, um mbm-Dateien (Multi BitMap) für die Ikons der zu entwickelnden Applikation zu erzeugen. Diese können aus .jpg oder .bmp - Dateien, die eingelesen werden, erzeugt oder selber in dem integrierten Editor erstellt werden. Sind diese Dateien später nicht vorhanden, werden Standardicons angezeigt. Sie sind damit also optional, gehören aber "zum guten Ton".

[V] Stand 4/2002

- AIFbuilder - Der Applikation Information File Builder wird benötigt, um symbianspezifische Dateien zu erzeugen, die später von *makesis* benötigt werden.
- makesis - mit diesem Kommandozeilenprogramm werden die erzeugten Komponenten zu einer `.sis`-Datei zusammengefügt, die als Installationsdatei auf dem Communicator 9210 dient und - ähnlich einer Applikationsinstallation unter Windows - mit Hilfe derer die Programminstallation auf dem Communicator 9210 durchgeführt wird.
- Emulator für den Communicator 9210 - Der Emulator für den Communicator 9210 liegt in zwei Versionen vor, eine Debug- und eine Releaseversion. Diese unterscheiden sich von der Handhabung nicht. Die Debugversion erzeugt und speichert jedoch zusätzlich Debuginformationen. Auf dem Emulator können Applikationen für den Communicator 9210 simuliert werden, ohne dass sie in das Gerät übertragen werden müssen.

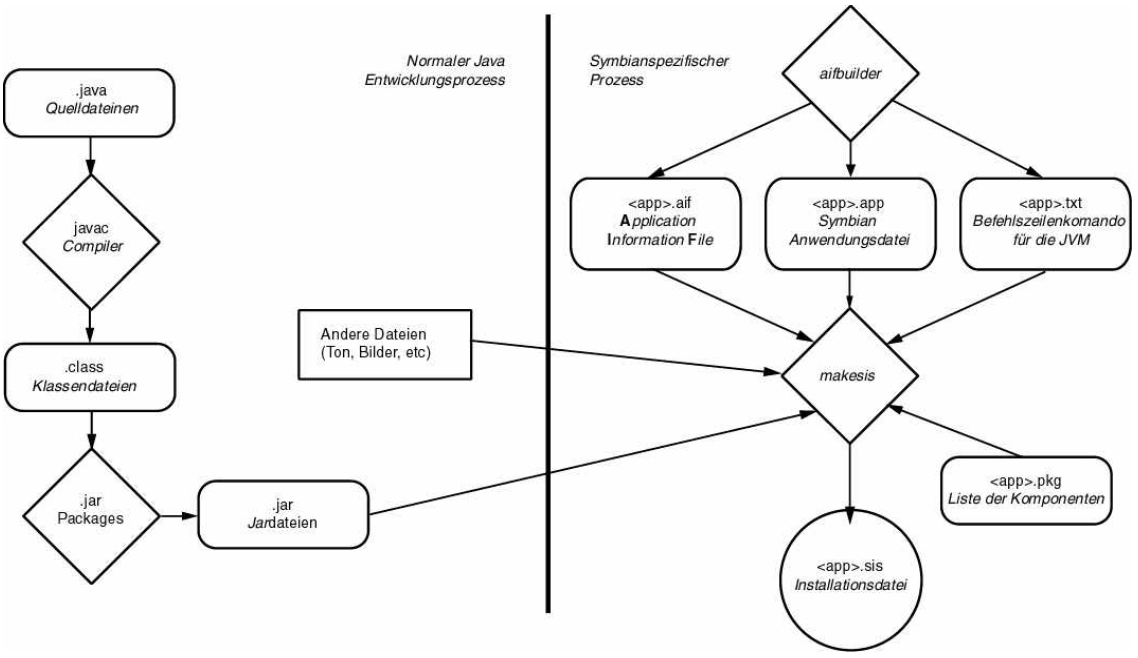

Abbildung 2. Java Development Diagramm
Quelle: Nokia [nokia]

Mit folgender Konfiguration plus einem Texteditor kann eine Applikation entwickelt werden:

Mit einem Texteditor wird eine `.java` - Datei erstellt. Diese wird mit *javac*, dem Java Compiler, kompiliert. Bis hier ist der Prozess noch nicht symbianspezifisch.

Die nun folgenden Schritte sind symbianspezifisch und benötigen daher die oben beschriebenen, im SDK enthaltenen Programme. Diese sind speziell für die Symbian Plattform und den Nokia 9210.

5.2.2 AIF Builder

Im ersten Schritt wird der AIF Builder gestartet.

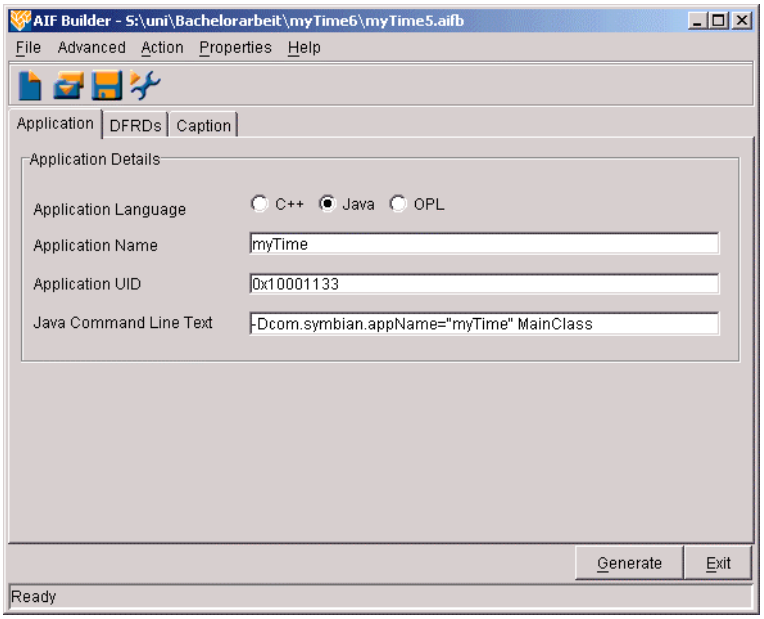

Abbildung 15: AIF Builder – Reiter Applikation

Auf dem ersten Reiter des AIF Builders, *Application*, wird festgelegt, von welchem Typ die Applikation ist, wie die Applikation heißt, welche UID (unicue identifier) sie bekommt und mit welchen Argumenten sie gestartet wird. Für meine Applikation wählte ich Java, sie bekam den Namen *myTime*, die UID 0x10001133 und als Starargumente `-Dcom.symbian.appName="myTime" MainClass`. Die UID ist eine 32-Bit Nummer, die gleichzeitig laufende Applikationen für das Betriebssystem unterscheidbar macht. Diese erhält man von Symbian[sym] auf Anfrage zugewiesen. Für den Testbetrieb übernahm ich jedoch die UID aus der Dokumentation und änderte die letzten beiden Ziffern jeweils um einen Zähler nach oben, was ebenfalls funktionierte.

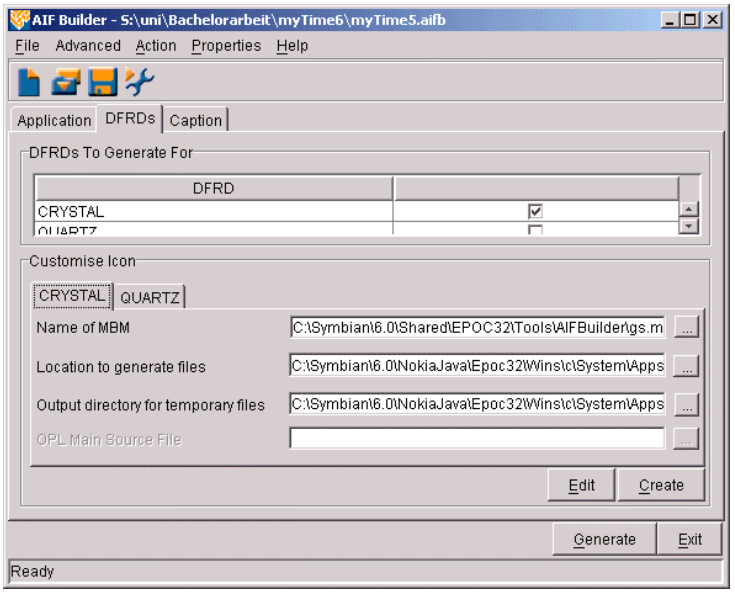

Abbildung 16 : AIF Builder – Reiter *DFRDs*

Als nächster Schritt ist der Reiter mit dem Namen *DFRDs* aus zu füllen.

5.2.3 AIF Icon Designer

Um den *DFRDs* Reiter ausfüllen zu können, muss eine mbm-Datei (Multi MiMap Datei) erzeugt werden. Diese ist eine Datei, die die Icons für die Applikation im Format 25x20 und 64x50 Pixel enthält. Dazu wird aus dem AIF Builder heraus der AIF Icon Designer gestartet. Aus diesem wiederum lässt sich problemlos ein .jpg-File für beide Auflösungen einlesen und das Ergebnis als .mbm abspeichern.

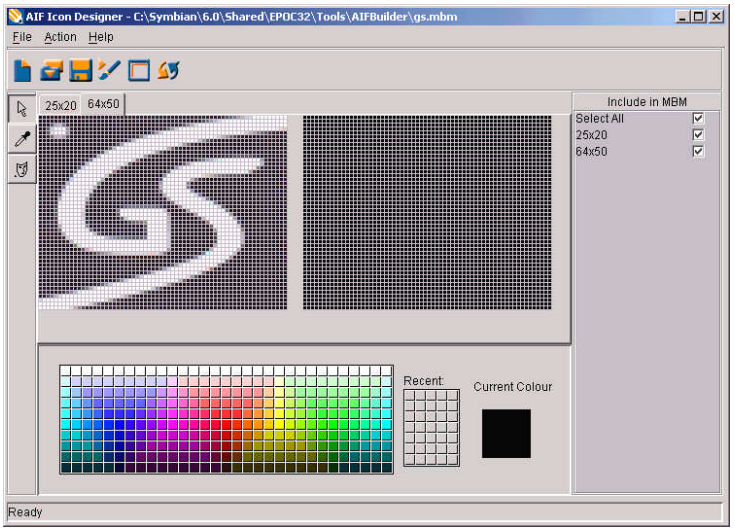

Abbildung 17: AIF Icon Designer

Hat man die .mbm-Datei erzeugt, trägt man im Reiter DFRDs, den Ort dieser Datei unter *Name of MBM* ein. Des weiteren wählt man auf diesem Reiter *CRYSTAL* aus, da auf dem Communicator 9210 die CRYSTAL-Version des Betriebsystems Epoc läuft. Auf diesem Reiter, wird das Verzeichnis für die Dateien eingetragen, die erzeugt werden sollen. Hier bietet es sich an, diese in einem Verzeichnis erzeugen zu lassen, das im Verzeichnispfad des Emulators liegt. Dies deshalb, damit man später die Applikation sofort im Emulator laufen lassen kann, ohne sie erst an eine andere Stelle kopieren zu müssen.

Nach nun erfolgter Konfiguration werden mit dem Knopf *Generate* die Dateien

- myTime.app
- myTime.txt
- myTime.aif
- myTime.rss

erzeugt.

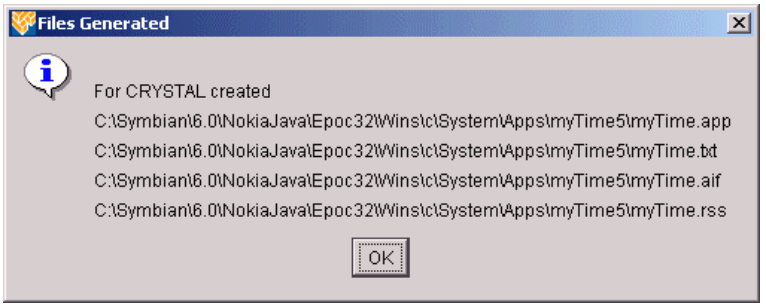

Abbildung 18: Diese Dateien werden vom AIF-Builder generiert

Oben gelistete Dateien sind als solche in Verbindung mit den kompilierten Java-Dateien und der mbm-Datei lauffähig. Um die Applikation im Emulator starten zu können werden sie manuell in das Verzeichnis `c:\Symbian\6.0\NokiaJava\epoc32\wins\c\System\apps\myTime` kopiert. Nun ist im Extras Menü des Communicator 9210 das erstellte Icon für *myTime* zu sehen und die Applikation startet, wenn das Icon ausgewählt und mit *Öffnen* geöffnet wird.

Um die Applikation auf dem Communicator 9210 zu starten, müssen die Dateien in die Systemverzeichnisse `c:\System\apps\myTime` oder `d:\System\apps\myTime` kopiert werden, wobei das Laufwerk `c` den internen Speicher des Communicator 9210, und das Laufwerk `d` die Speicherkarte des Communicator 9210 darstellt. Die Dateien werden entweder über das mitgelieferte serielle Kabel und die Communicator 9210 eigene Software an die gewünschte Stelle kopiert oder mittels eines MMC-Kartenlesegerätes auf die Speicherkarte kopiert.

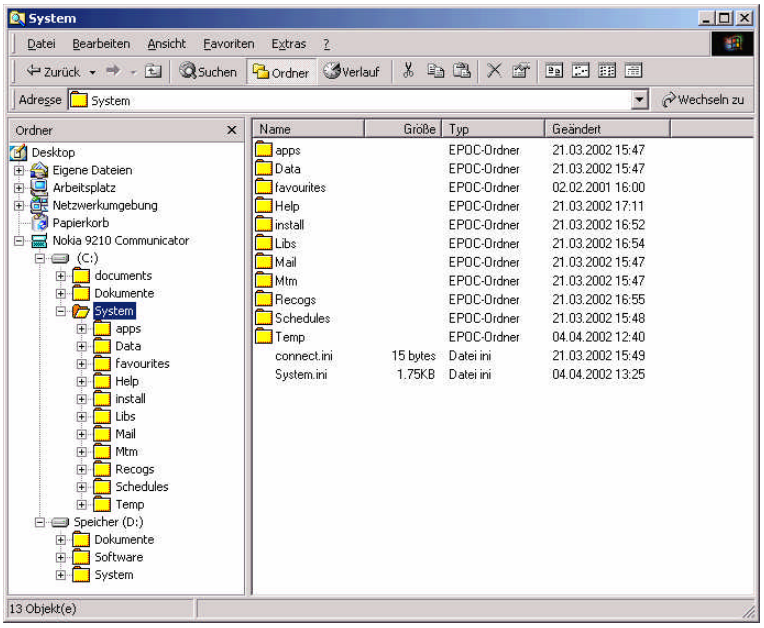

Abbildung 19: Dateisystem im Windows Explorer integriert

Für eine professionelle Softwareverteilung ist es von Vorteil, wenn der Benutzer eine einzige Installationsdatei bekommt, in der alle Dateien und Informationen, die zur Installation benötigt werden, zusammengefasst sind. Genau diese Installationsdatei wir mit Hilfe von *makesis* erstellt. *Makesis* benötigt eine Datei gleichen Namens wie die Applikationsdateien mit der Endung `.pkg`. Diese ASCII Datei enthält Informationen über Sprachversion, UID, Applikationsname und eine Liste der Dateien, welche in die `.sis`-Datei eingebunden werden und wie sie an ihrem Ziel heißen sollen.

Ist diese `.pkg` - Datei erstellt kann mit Hilfe des Befehls `makesis myTime.pkg` die Installationsdatei `myTime.sis` erstellt werden.

Diese kann nun wie oben beschrieben in das Dateisystem des Emulators integriert werden und über den das Menü *Software installieren* auf dem Emulator installiert und gestartet werden.

Für die Installation auf dem Communicator 9210 wird die Installationsdatei auf eine der zwei oben beschriebenen Arten auf selbigen übertragen und, wie im Emulator auch, installiert. Sie ist allerdings nicht ohne weiteres lauffähig. Es müssen erst die Symbian eigenen Java-Klassen `cawt.jar` und `classes.zip` in das System integriert werden. Dies geschieht, indem man diese beiden Dateien in das Systemverzeichnis `!:\system\Java\ext\` kopiert. Dabei spielt es keine Rolle, ob Laufwerk c (Speicher des Gerätes) oder Laufwerk d (Speicherkarte) gewählt wird. Dieses Problem kann umgangen werden, indem man diese beiden Klassen in die `.sis` - Datei einbindet.

Zur Implementierung von *myTime* wurden noch weitere Komponenten verwendet, um den Entwicklungsprozess zu vereinfachen und zu beschleunigen:

5.2.4 JBuilder5

Anstelle eines Texteditors zur Erstellung der Java-Quelldateien wurde der JBuilder5 von Borland als Java-Enwicklungsumgebung verwendet. Die personal Edition kann von Borland kostenlos von der Internetseite bezogen werden[jbuil]. Dazu ist eine Registrierung bei Borland[borl] notwendig.

Nach der Installation, muss der JBuilder auf das SDK 1.1.8 umgestellt werden und die Symbianklassen in den Klassenpfad eingebunden werden. Mit dieser Konfiguration können nun Javaklassen erzeugt werden, die für den weiteren Entwicklungsprozess geeignet sind.

Auf der Internetsite des Forum Nokia ist seit ende 2001 die *Nokia Developer's Suite for PJAE* (kurz: Developer' Suite) erhältlich, die den Entwicklungsprozess weiter vereinfacht. Diese wurde bei der Implementierung von *myTime* eingesetzt. Sie vereinfacht die Schritte ab der Kompilierung bis hin zur Installation auf dem Communicator 9210. Die Developer's Suite kann als einzelne Anwendung installiert werden, oder in den JBuilder integriert werden.

5.2.5 Nokia Developer's Suite for PJAE

Bei der Entwicklung von *myTime* wurde die Developer's Suite in den JBuilder integriert. Dabei wird dem JBuilder ein Menüunterpunkt in dem Verzeichnis *Tools* hinzugefügt.

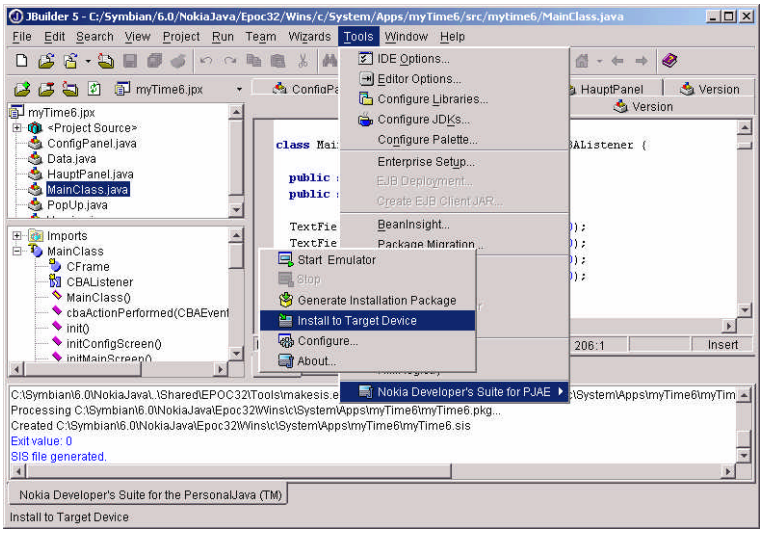

Abbildung 20: In den JBuilder eingebundenes Menü

Über dieses Menü können Einstellungen der Developer's Suite vorgenommen werden, der Emulator gestartet werden, Installationsdateien (`.sis`-Dateien) erzeugt werden und die fertige, oder zum testen bereite, Applikation im Communicator 9210 installiert werden. Die Konfiguration der Developer's Suite wird zusammen mit dem JBuilder-Projekt automatisch abgespeichert.

Für *myTime* wurde die Developer's Suite folgendermaßen konfiguriert:

5.2.5.1 Reiter "Emulator"

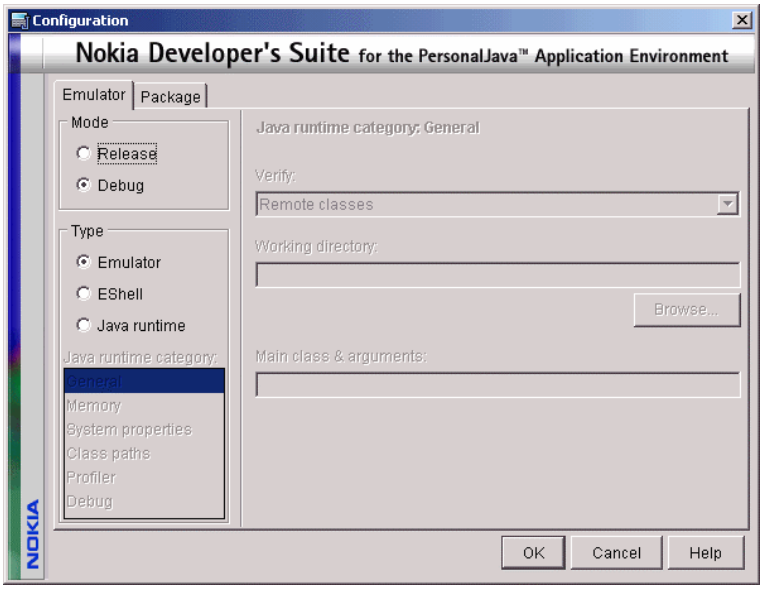

Abbildung 21: Screenshot: Developer's Suite – Reiter Emulator

Folgende Einstellungen wurden in diesem Reiter vorgenommen:

- Mode: Release
- Type: Emulator

5.2.5.2 Reiter "Package"

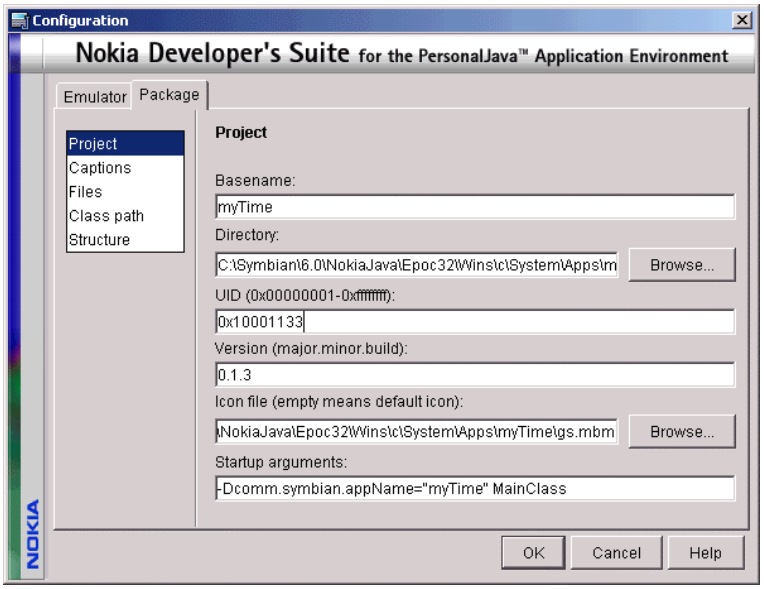

Abbildung 22: Screenshot: Developer's Suite – Reiter *Package*

Für *myTime* wurde der Reiter "Package" folgendermaßen konfiguriert:

- Basename: myTime – dies ist der Name, der für die Erzeugung der Dateien verwendet wird. Er hat jedoch keinen Einfluss auf den Namen der Applikation
- Directory: `C:\Symbian\6.0\NokiaJava\Epoc32\Wins\c\System\Apps\myTime` - Dies ist das Verzeichnis, in welches der JBuilder auch die fertigen Klassen schreibt.
- UID: 0x10001133 - ein *Unique Identifier*, der an den letzten beiden Stellen um jeweils eine Ziffer von der in der Dokumentation abweicht.
- Versionsnummer: Jeweils die Aktuelle Versionsnummer
- Icon file: Pfad und Name der `.mbm` - Datei welche die Icons für die Anwendung enthalten
- Startup Arguments: `-Dcomm.sysmbian.appName="myTime"_MainClass` Argumente für das Starten der Anwendung: Applikationsname – *myTime*, zu startende Java Klasse – *MainClass*

5.2.5.3 Reiter „Caption“

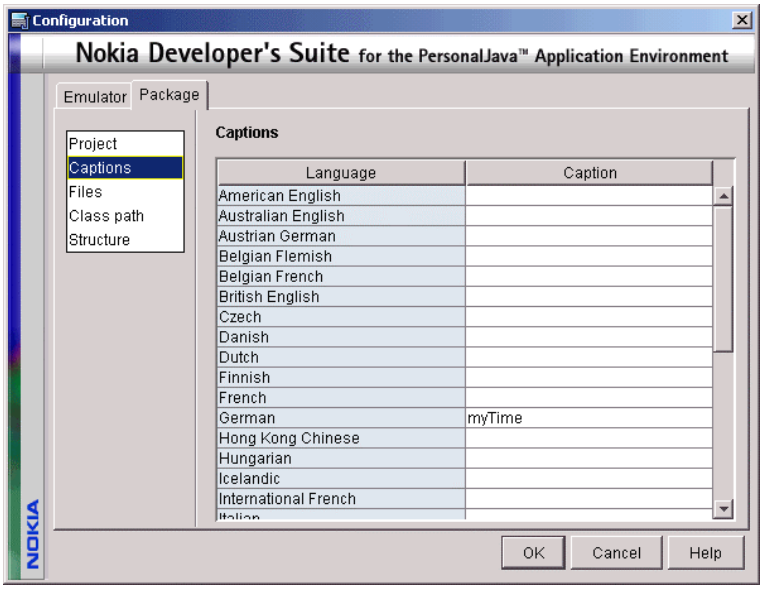

Abbildung 23: Developer’s Suite – Reiter Caption

Der Reiter „Caption“ wurde folgendermaßen konfiguriert:

- *myTime* als Eintrag bei „German“. Dieser Eintrag wird bei der Installation berücksichtigt, wenn die Sprachen ausgewählt werden. *myTime* erscheint bei der Sprachauswahl „German“ als Titel der Anwendung.

5.2.5.4 Reiter „Caption“

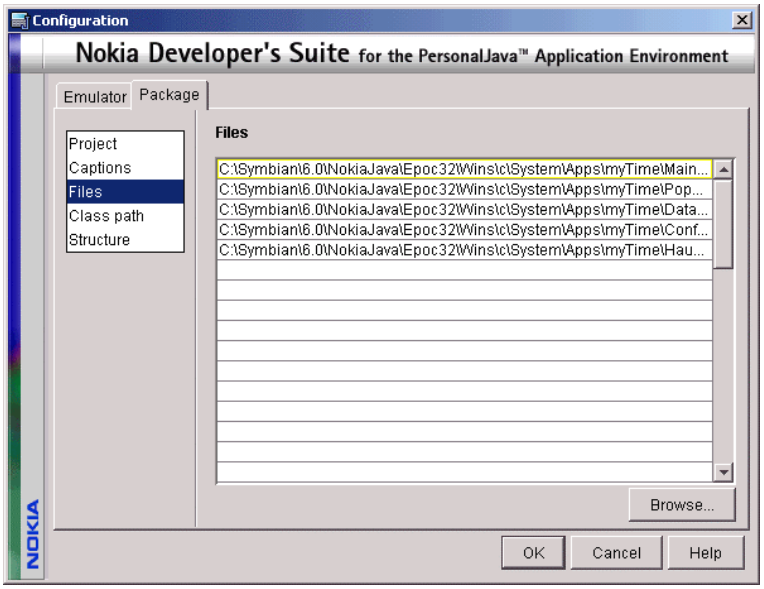

Abbildung 24: Developer’s Suite – Reiter *Files*

Als Files wurden folgende Dateien eingetragen, die in die Applikation eingebunden werden:

- MainClass.class
- PopUp.class
- Data.class
- ConfigPanle.class
- HauptPanel.class
- Version.class

5.2.5.5 Reiter “Class Path”

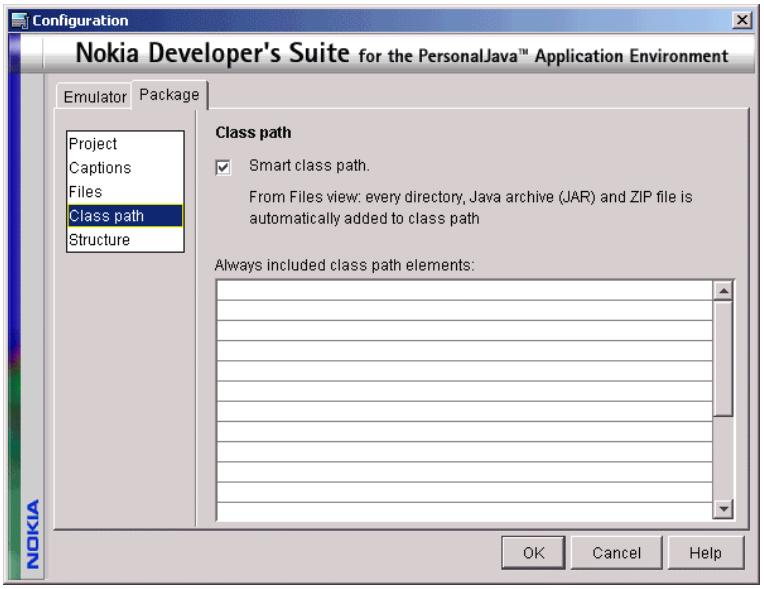

Abbildung 25: Developer’s Suite – Reiter *Class path*

Wir die Option „Smart class path“ gesetzt, wird der Klassenpfad automatisch so angepasst, dass die Dateien, die im Reiter „Files“ definiert sind, eingeschlossen werden.

5.2.5.6 Reiter „Structure“

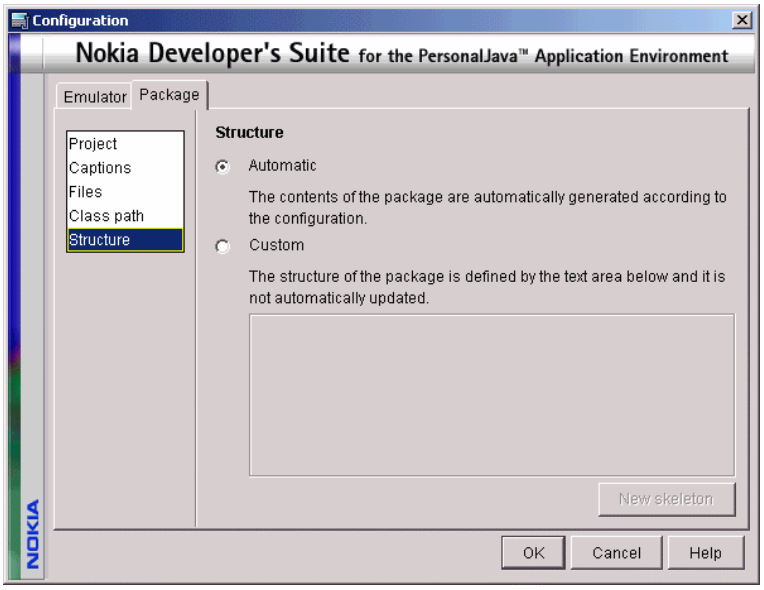

Abbildung 26: Developer’s Suite – Reiter *Structure*

Hier gilt gleiches wie bei dem Reiter „Class path“: Wird „automatic“ gewählt wird entsprechend den gewählten Dateien im Reiter „Files“ die Struktur angepasst.

5.2.6 Probleme bei der Implemeniterung

Folgende Probleme traten im Zusammenhang mit dem Emulator bei der Entwicklung von *myTime* auf:

Der Emulator benötigte über eine Minute, vom Start bis zur Bereitschaft, Programme ausführen zu können. Wird das ausgeführte Java-Programm gestartet, kann nur diese Version getestet werden. Bei einer neuen Kompilierung werden die Änderungen nicht erkannt und der Emulator muss neu gestartet werden. Abhilfe wurde dadurch geschaffen, dass auf einmal mehrere Instanzen des Emulators gestartet wurden, die im Hintergrund aufstarteten und dann nacheinander benutzt werden konnten.

Nach dieser Installation und Konfiguration, konnte der Prozess von Codeschreiben bis zum Testen im Emulator auf unter einer Minute reduziert werden - Vorraussetzung für eine effektive Entwicklung.

5.2.7 Konfiguration des mobilen Endgerätes

Als Testumgebung wurde der Nokia Communicator 9210 mit der Firmware 3.62 verwendet. Diese ist das zweite Release des Betriebsystems für den Communicator 9210. Datenaustausch mit dem PC wurde über das mitgelieferte Serielle Kabel und einem MMC Kartenlesegerät erreicht. Installiert wurde die Java VM für Symbian (Installation des Paketes `java.sis` von der Seite des *Forum Nokia*[fnok]) und manuelle Installation der Pakete `cawt.jar` und `javaphone.jar` im Verzeichnis `d:\System\Java\ext`.

Die Java VM belegt etwa 2,5 MB auf der Speicherkarte oder im RAM des Communicator 9210. Die Java VM wird erst geladen, wenn das Java-Programm aufgerufen wird. Dies verlangsamt den Startprozess, schont jedoch Speicherressourcen.

5.2.8 Funktionsweise von myTime auf Classmark 2

Die Beispielanwendung *myTime* ist ein mobiles Zeiterfassungssystem, implementiert auf dem Nokia Communicator 9210. Sie soll die Anwendungsentwicklung auf dem Communicator 9210 und damit einem Classmark 2 Gerät illustrieren.

In der Einleitung beschriebenes (Seite 31, Kap. 5) Szenario ist bei dieser Implementierung zu Grunde gelegt worden:

Eine Person, die mobil Zeit erfassen möchte benutzt einen Communicator 9210. Die Person möchte die Zeit, die sie für einen bestimmt Sache benötigt, erfassen und die so erfasste Zeit zur Weiterverarbeitung an ein System übermitteln. In dieser Implementierung kann zwischen SMS und http gewählt werden.

Abbildung 27: *myTime*, wie sie im Display nach der Installation erscheint

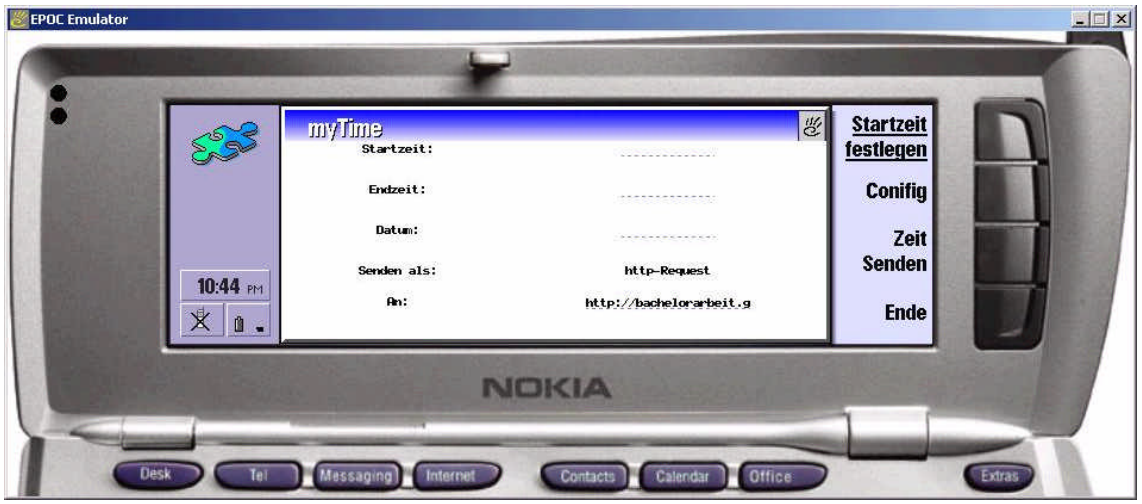

Abbildung 28: *myTime* nach dem Start

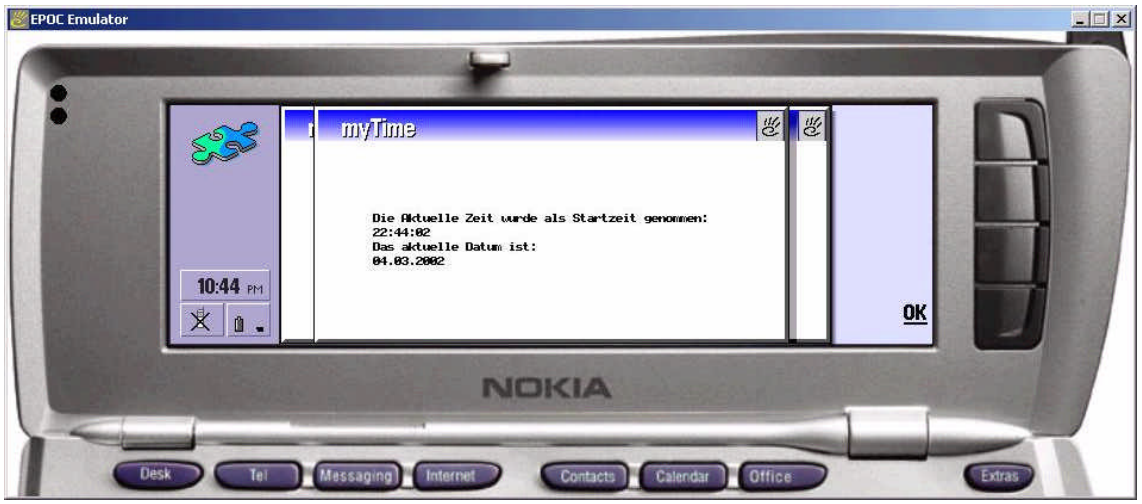

Abbildung 29: PopUp nach genommener Anfangszeit

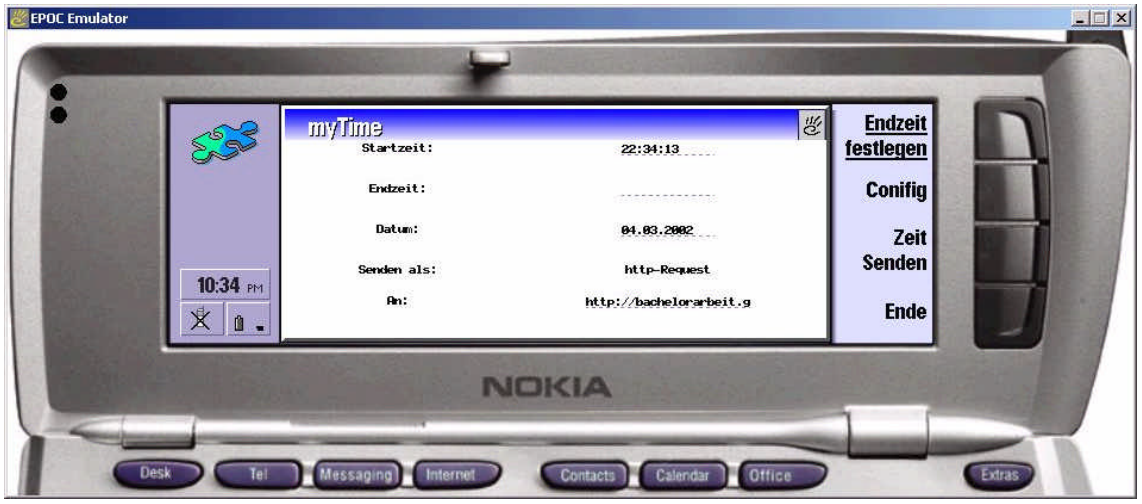

Abbildung 30: Nochmals der erste Screen, nachdem das PopUp bestätigt wurde. Die Zeit kann manuell verändert werden

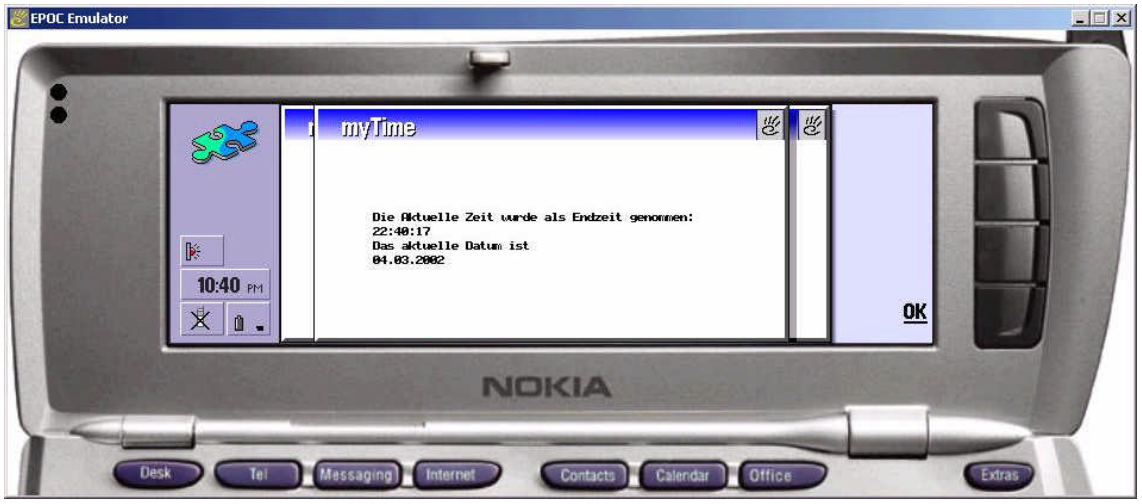

Abbildung 31: PopUp nach genommener Endzeit

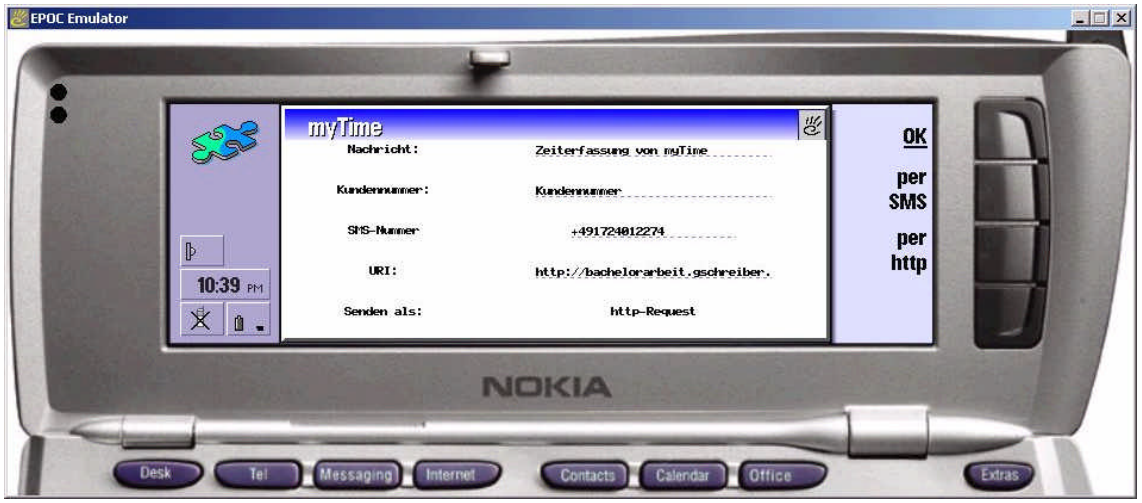

Abbildung 32: Konfigurationsbildschirm mit „http-Request“ ausgewählt

1. Zeile: Hier kann eine individuelle Nachricht eingegeben werden, die hinten an die Nachricht gehängt wird. Hier ist auch eine Kundennummer denkbar oder eine Rechnungsnummer.
2. Zeile: Kundennummer: Hier kann eine Kundennummer eingegeben werden, die dafür gedacht ist, das das System auf der Serverseite die erfassten Daten einem bestehenden Datensatz zuordnen kann.
3. Zeile: Nummer der SMS an den die Daten gesendet werden sollen (bei „Senden als“: SMS)
4. Zeile: URL an den die Daten gesendet werden sollen. Diese Zelle wird ausgelesen und verwendet, wenn bei „Senden als“ : „http-Request“ eingestellt ist.
5. Zeile: Senden als (entweder SMS oder http-Request). Springt um, wenn der entsprechende Button gedrückt wird.

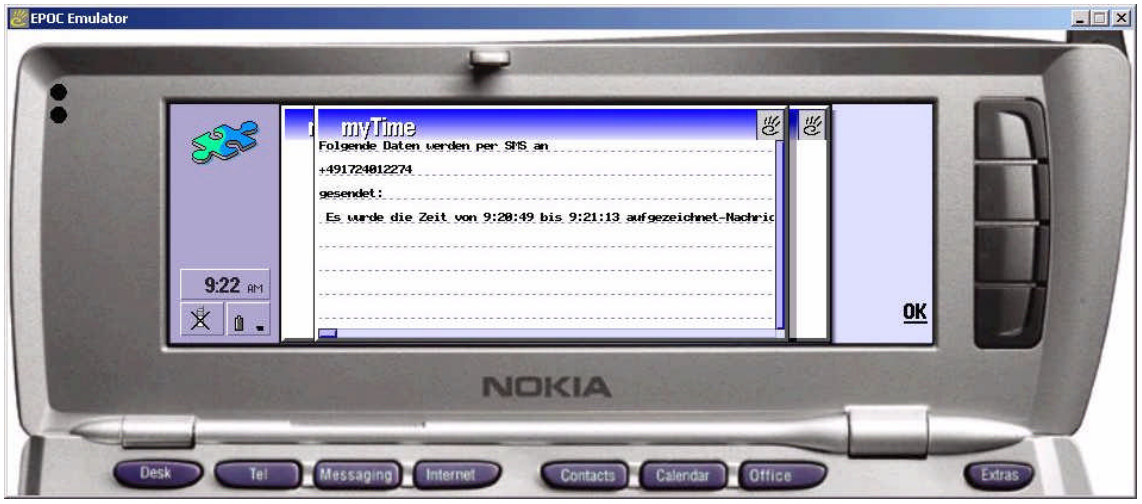

Abbildung 33: Bestätigung für die gesendete SMS

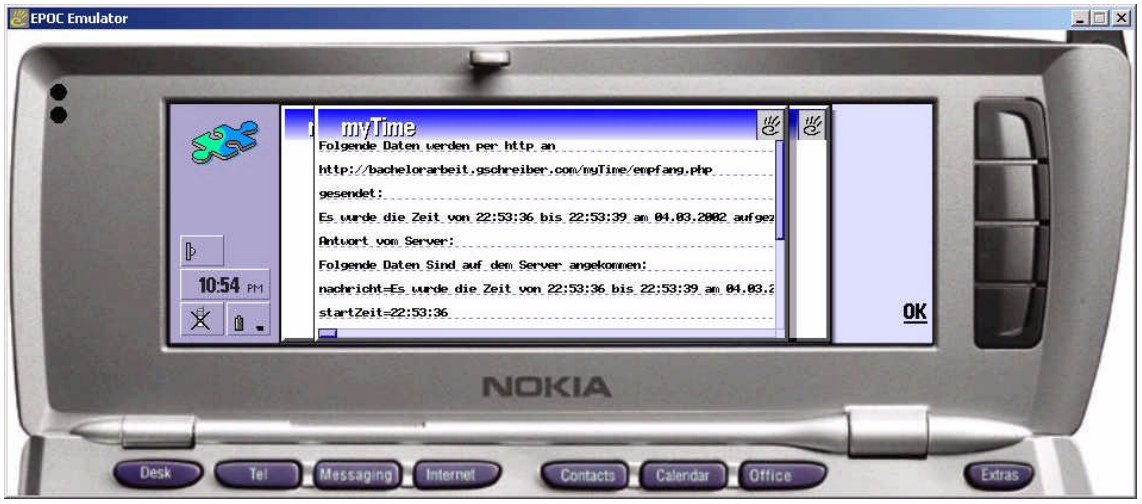

Abbildung 34: Bestätigung für per http gesendete Daten und die Bestätigung des Server (ab der Zeile „Antwort vom Server“)

Die Datei http://www.bachelorarbeit.gschreiber.com/myTime/empfang.php ist auf der Serverseite für die Verarbeitung der Daten, die als http-Request gesendeten werden, verantwortlich.

In der vorliegenden Implementierung wird eine Email mit den empfangenen Daten generiert und verschickt. Denkbar wäre an dieser Stelle auch ein Datenbankeintrag oder eine anderweitige Verarbeitung der ankommenden Daten. Da die Serverseite in der vorliegenden Arbeit jedoch nicht Schwerpunkt ist, wird dies deshalb nicht weiter ausgeführt.

5.3 myTime mit Classmark 3

Um *myTime* mit Classmark 3 zu realisieren, wurde eine Entwicklungsumgebung von Nokia und Borland ausgewählt: *JBuilder5 Mobile Set Nokia Edition*. Diese wurde ausgewählt, weil zum einen bereits von der Classmark 2 Implementierung Erfahrungen mit dem JBuilder und der integrierten Entwicklungsumgebung vorhanden waren, zum Anderen um einen unmittelbaren Vergleich zwischen den beiden Entwicklungsumgebungen zu erhalten.

Das SDK basiert auf der MIDP Referenzimplementierung des Wireless Toolkit von Sun.

Da derzeit die Classmark 3 Geräte sich erst auf dem Markt etablieren müssen und noch nicht allzu verbreitet sind[VI] und kein Classmark 3 Gerät zu Testzwecken vorlag, wurde für den Test der Implementierung lediglich ein Emulator auf einem Windows 2000 Betriebssystem verwendet.

Abbildung 3. Borland JBuilder5 mit Nokia Mobile Set

[VI] Stand: April 2002 nach der CeBIT

5.3.1 Konfiguration der Entwicklungsumgebung

Die Entwicklungsumgebungskonfiguration war die gleiche Hard- und Software, wie bei der Entwicklung für Classmark 2:

PC mit Pentium III, 512MB RAM, CD-Rom, 40GB HD ATA 100, Windows 2000 deutsch mit Sevice Pack 2, JBuilder 5 Personal Edition.

Der JBuilder5 – Personal Edition kann von der Borlandwebseite kostenlos unter Voraussetzung der Registrierung bezogen werden[jbmob]. Nach einer Menügeführten Installation präsentiert sich der JBuilder5 im gewohnten Bild.

Um mit dem MIDP-JDK arbeiten zu können, muss nun das Mobile Set als JDK unter „Tools“ -> „Configure JDKs“ im JBuilder5 eingebunden werden. Ist dies geschehen, kann mittels „File“ -> „New“ direkt eine neue MIDlet Suite anlegen.

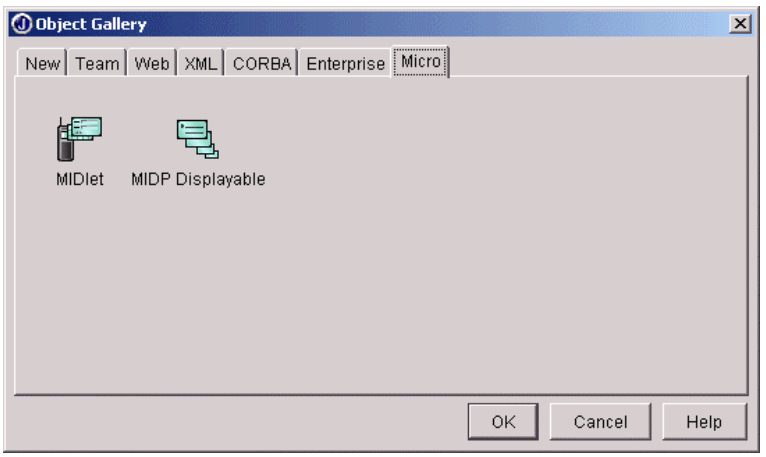

Abbildung 35: Aus dem JBuilder direkt ein neues MIDlet erzeugen

Nun kann man entweder die Werkzeuge benutzen, die dem JBuilder neu hinzu gefügt wurden oder den Quelltext manuell eingeben.

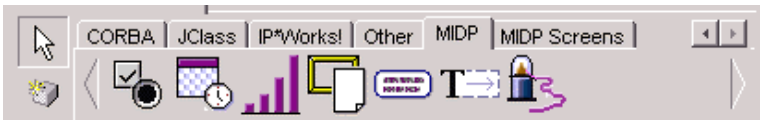

Abbildung 36: Reiter “MIDP” im JBuilder5 mit Tools für MIDP-Elemente

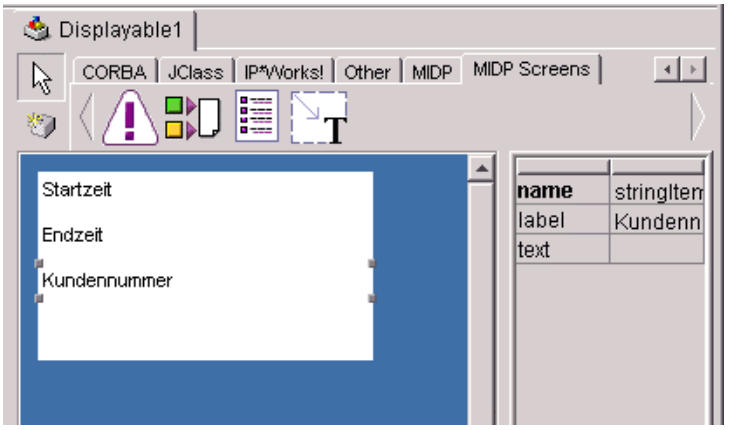

Abbildung 37: Visuelle Entwicklung im "Designer" des JBuilder5

Die weitere Entwicklung einer Applikation gestaltet sich recht einfach und unkompliziert, ganz anders als bei dem Communicator 9210.

In den Projekteinstellungen („Project" -> „Project Properties" -> Reiter „Run") wurde bei der Installation des *Mobile Set* für die Entwicklung von MIDlets ein weiterer Reiter namens „MIDlet" hinzugefügt. Dort kann die zu startende Klasse für dieses Projekt festgelegt werden.

Hat man den Quelltext soweit zusammengestellt, dass man für einen Test bereit ist, lässt sich nun alles weitere mit einem Knopfdruck („F9") starten. Diese Aktion startet den Compiler und ruft anschließend den Emulator mit der kompilierten Datei auf. Alternativ kann man mit der rechten Maustaste die Datei anwählen und mit „Micro Run" (auch vom Mobile Set während der Installation hinzugefügt) die Datei starten.

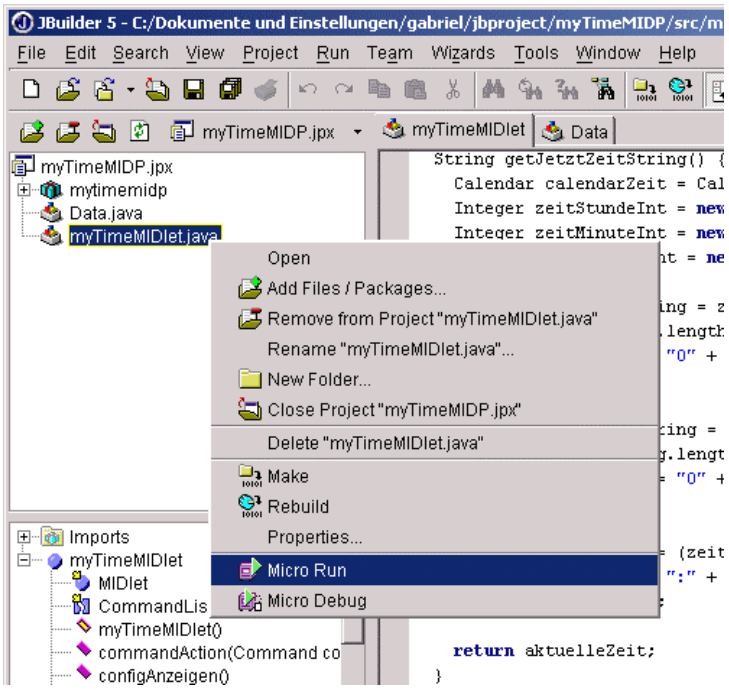

Abbildung 38: Datei starten mit "Micro Run"

Als Ergebnis erscheint der Emulator des Mobilen Gerätes (in vorliegenden Fall ein Nokia 6310i) mit laufendem MIDlet.

Abbildung 39: Emulator Nokia 6310i mit laufendem MIDlet *myTime* nach dem Start

5.3.2 Entwicklung und Struktur von myTime auf MIDP

Die graphische Anzeige der MIDlets wird über sog. Displayable–Screens gesteuert. Diese sind "Minnibildschirmseiten", von denen das MIDlet jeweils eine anzeigen kann. Ihnen werden die möglichen Elemente, wie Form, Alert, TextBox, List, die von der Klasse „Screen" stammen hinzugefügt und jeweils mit dem aktiven Displayable angezeigt. Displayables können eigene ComandListener besitzen, die auf einen begrenzen Satz von Kommandos hören. Diese Kommandos sind auf das begrenzte User Interface zugeschnitten und bestehen aus:

- BACK
- CANCEL
- EXIT
- HELP
- ITEM
- OK

- SCREEN
- STOP

Diese Kommandoklasse kann nicht erweitert werden. Da es sich um ein hohes Abstraktionsniveau handelt, ist es der jeweiligen Implementierung der Java-VM überlassen, wie die Kommandos implementiert werden. So kann beispielsweise ein OK-Knopf das drücken auf ein Einstellrädchens sein oder ein Tastendruck auf eine Taste unterhalb der Anzeige.

Für *myTime* wurden sechs Displayables eingesetzt:

- List (Typ List)
  - Enthält eine Auswahlliste aller Aktionen, die zur Auswahl stehen: Startzeit (erfassen), Endzeit (erfassen), Zeit Senden, Senden an... (konfigurieren), Kundennummer (konfigurieren)
- StartZeitAlert (Typ Alert)
  - Benachrichtigt den Benutzer darüber, welche Startzeit erfasst wurde
- EndZeitAlert (Typ Alert)
  - Benachrichtigt den Benutzer darüber, welche Endzeit erfasst wurde
- nachrichtAlert (Typ Alert)
  - Gibt die Rückmeldung im Zusammenhang mit der gesendeten Zeit an den Benutzer aus.
- textBoxURL (Typ TextBox)
  - Zur manuellen Einstellung einer Empfänger URL, an die die Daten gesendet werden.
- textBoxKundennummer (Typ TextBox)
  - Zur Eingabe einer Kundennummer, die mitübertragen werden soll.
  -

5.3.3 Funktionsweise von myTime auf Classmark 3

Nach dem Starten von *myTime* erscheint als erstes das Auswahlmenü in dem man zwischen den Aktionen auswählen kann:

Abbildung 40: *Startzeit* - erfasst die aktuelle Zeit

Abbildung 41: *Startzeit* - erfasst die aktuelle Zeit

Abbildung 42: *Endzeit* - erfasst die aktuelle Zeit

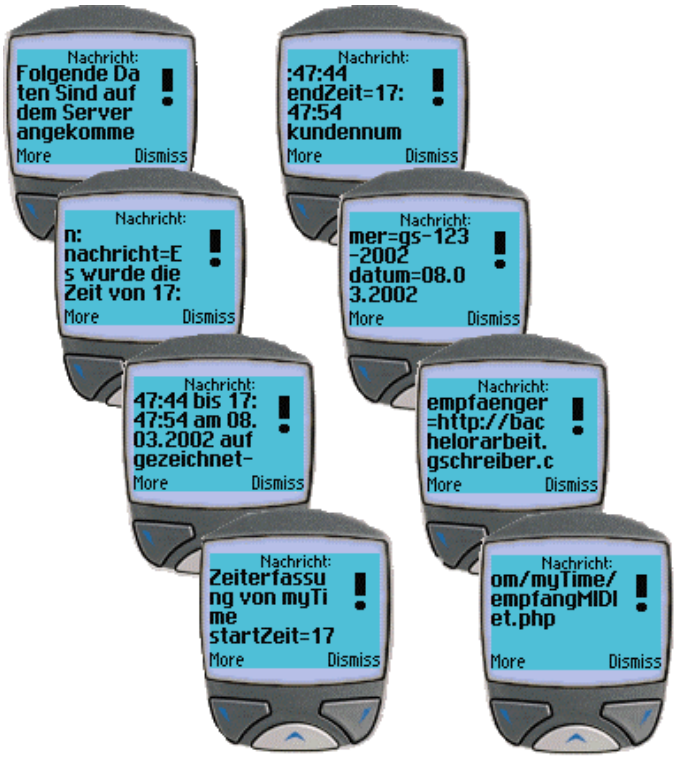

Abbildung 43 : *Zeit Senden*: öffnet eine http Verbindung
und sendet die Zeit an den Server

Abbildung 44: *Senden An...* - Unter diesem Menüpunkt kann man festlegen, an welche Server URL die Daten geschickt werden sollen

Abbildung 45: *Kundennummer* - Unter diesem Menüpunkt eine Kundennummer eingeben werden

6 Erfahrungen bei der Implementierung von myTime

6.1 myTime auf Classmark 1

Es ist nicht möglich, auf Classmark 1 (WAP) SMS zu verschicken. Des weiteren ist es, wie oben bereits erwähnt, nicht möglich, die Zeit automatisch aus zu lesen.

In der Praxis gestaltet sich die Erstellung von WML mit den entsprechenden Werkzeugen recht einfach, bietet jedoch kaum mehr Funktionalität als html. Verglichen mit html ist WAP jedoch schwieriger zu erstellen, da es nicht wie html fehlertolerant ist. Vielmehr muss es erst kompiliert werden, bevor es angezeigt werden kann. Diese Eigenschaft wurde WAP hinzugefügt, damit nur die Daten über das (drahtlose) Netz gesendet werden müssen, die wirklich nötig sind.

Auch gibt es keine WISWYG–Editorn auf dem Markt, wie sie für html weit verbreitet sind.

Dadurch dass es im Gegensatz zu Classmark 2 und Classmark 3 so gut wie nicht möglich ist, Programme zu erstellen, die auf dem Telefon laufen, sowie aufgrund der beschränkten Anzeigemöglichkeit des Displays und dem oben beschriebenen Aufwand zur Erstellung von wml, ist es nicht sehr attraktiv, für Classmark 1 zu entwickeln.

6.2 myTime auf Classmark 2

In der Praxis hat bei der Implementierung von *myTime* die Klassenbibliothek `cawt.jar` gefehlt. Diese enthält die GUI Elemente für den Communicator, die für die Darstellung erforderlich sind. Erst nach deren Kopie auf das Endgerät lief *myTime*.

Eine Installation von *myTime* auf einem zweiten Gerät, mit einer Testperson, der nicht mit den Entwicklungsdetails vertraut war, erwies sich für diese als schwierig. Um die Datei `cawt.jar` in das richtige Verzeichnis zu bekommen muss zuerst die Anzeige im Dateimanager unter Einstellungen „Systemverzeichnis anzeigen“ auf „Ja“ gestellt werden. Erst dann ist es möglich, `cawt.jar` in das Verzeichnis `!:\System\Java\ext` zu verschieben. Das ist die Vorrausetzung für das Funktionieren von *myTime*.

Die Ladezeit der JavaVM beträgt etwa drei bis vier Sekunden und benötigt etwa 1MB freien Arbeitsspeicher. Ist dieser nicht vorhanden, wird die Applikation nicht gestartet. Die Zuverlässigkeit der Programmausführung hängt also davon ab, ob und wie viele andere Programme geöffnet sind. Selbst ein simples HelloWorld-Programm wird oft nicht gestartet. Mit anderen Worten ausgedrückt: die Zuverlässigkeit ist unzureichend für eine professionelle Softwareentwicklung.

Anwendungen für den Nokia Communicator 9210 zu erstellen gestaltet sich nach der Installation und Beschaffung der Entwicklungsumgebung anfangs einfach und überschaubar, da auf bekannte Java–Klassen zurückgegriffen werden kann. Will man jedoch komplexere Anwendungen realisieren, können diese nicht einfach portiert werden, sondern erfordern die Verwendung der symbianspezifischen Entwicklungswerkzeuge und Klassen um die Besonderheiten des Gerätes ausschöpfen zu können. Dies zusammen mit der unzuverlässigen VM und der notwendigen Nachinstallation der symbianeigenen Klassen macht es unmöglich, Applikationen zu entwickeln, die für den Benutzer „out of the box“ laufen.

Die Unterstützung seitens Nokia im Internet, sowie gute Dokumentation sind eine hilfreiche Unterstützung.

6.3 myTime auf Classmark 3

Es war nicht möglich, basierend auf Classmark 2 SMS zu verschicken, deshalb musste auf SMS Funktionalität bei dieser Implementierung verzichtet werden.

Des weiteren steht im MIDP das Paket `java.net` und damit die Klasse URL nicht zur Verfügung. Es war damit nicht möglich auf einen URLEncoder zurück zu greifen.

Mit dem vorhandenen Paket `javax.microedition.io` welches die Klasse `HttpConnection` enthält, gab es Probleme mit dem Schreiben auf eine URL. Deshalb wurden die Daten mit der http-GET-Methode übermittelt. Ob diese Probleme an der Implementierung des Emulators liegen, konnte nicht festgestellt werden.

Um dennoch die Daten kodieren zu können, wurde manuell ein Encoder zu der Applikation hinzugefügt (Methode `encode(String s)` in der Klasse `mytimemipd.Data`). Diese werden dann an den http- Querystring gehängt und auf diese Weise übermittelt.

In der Praxis gestaltet sich die Erstellung von MIDlets recht einfach und überschaubar. Die MIDP-Klassen sind gut dokumentiert und visuell ebenso gut in den JBuilder5 eingebunden. Dagegen steht jedoch ein sehr geringer Funktionsumfang.

Sobald mehr Geräte auf den Markt kommen, die das MIDP unterstützen, werden sicherlich mehr Anwendungen dafür entwickelt werden. Die Entwicklungsumgebungen sind ohne großen Aufwand erhältlich. Allerdings ist zu bezweifeln, ob die sehr eingeschränkten Möglichkeiten von Classmark 3 ausreichen, um ernsthafte Anwendungsentwicklung dafür attraktiv zu machen.

7 Fazit und Ausblick

Mit *MExE* wurde versucht, die Anwendungsentwicklung auf Mobilen Endgeräten zu standardisieren und zu klassifizieren. Generell ist zu hinterfragen, ob Standardisierung von Anwendungsentwicklung überhaupt sinnvoll und in der Praxis durchführbar ist. Zu viele Hersteller setzen auf proprietäre Lösungen und die Standards werden nur unzureichend eingehalten. Der Versuch mit MExE einen Standard zu schaffen wird vermutlich an der Javalstigkeit scheitern, denn Microsoft wehrt sich mit Händen und Füßen gegen Java und versucht mit seiner Marktmacht .net. durchzusetzen.

Classmark 1 (WAP) wurde, nach anfänglicher Euphorie der Hersteller, vom Markt nicht angenommen. Für Classmark 2 (PersonalJava) gibt es nach über zwei Jahren bislang nur ein Gerät, welches den Standard zwar unterstützt, jedoch mit der Implementierung von diesem nicht Alltagstauglich ist, so dass die meisten Anwendungen für dieses Gerät proprietäre Lösungen in C++ sind. Für Classmark 3 bestehen insofern die meisten Chancen, als dass nun immer mehr Geräte auf den Markt kommen, die Classmark 3 unterstützen, die Entwicklungsumgebungen sind am weitesten ausgereift, weil in bestehende Java Umgebungen integrierbar sind und die Implementierung ist am einfachsten zu bewerkstelligen. Auch die VM stellt einen gesunden Kompromiss aus Funktionalität und Anforderungen an die Ressourcen dar.

Abbildungsverzeichnis

Literatur und Linkliste

Literatur

Palm OS

[palos] Palm OS Programming Bible von Lonnon R. Foster, Hungry Minds, Inc, ISBN: 0764546767

[palpro] Das große Buch Palm- Programmierung. Schnell zum Ziel ohne Programmierkenntnisse, Christian Immler, DATA Becker, ISBN: 3815821215

[palpra] Palm Pilot Praxisbuch, Uwe Debacher, Franzis Feldkirchen Erscheinungsdatum: 2001, ISBN: 3772367763

Psion / Epoc

[psimo] Psion - Mobile Computing aus der Praxis, Lebe, Nöhbauer, Axel Baumgart u.a., wjr-verlag, ISBN: 3980722910 (England)

[sympro] Professional Symbian Programming: Mobile Solutions on the EPOC Platform, Martin Tasker, Jonathan Allin, Jonathan Dixon, Mark Shackman, John Forrest, ISBN: 186100303X

[psipo] Das PSION Power- Buch. Für Serie 5, 5mx PRO sowie revo, revo PLUS und netBook, Fredy Ott, Thomas Schmidt, Thomas Ullrich X.Media Verlag, ISBN: 3932888162

Windows CE

[ceeins] Pocket PC und Windows CE 3.0 für Einsteiger,Rainer Gievers ,BoD, Norderstedt, ISBN: 3831118779

[ceapp] Windows CE 3.0 Application Programming, Nick Gratten, Marshall Brain, Prentice Hall PTR, ISBN: 0130255920 (England)

Java Mobile

[japrom] Professional Java Mobile Programming, Bill Ray, Steve Atkinson, ua, Wrox Press Ltd, ISBN: 1861003897

[jamidp] Mobile Information Device Profile for Java 2 Micro Edition (J2ME): Professional Developer's Guide, Enrique Ortiz, Eric Giguere, John Wiley & Sons, Inc. Erscheinungsdatum: 7. September 2001, ISBN: 0471034657

Links

[ba] Webseite dieser Bachelorarbeit

http://bachelorarbeit.gschreiber.com

[mexef] MExE Forum

http://www.MExEforum.org/

[sun] Sun Microsystems

http://www.sun.com

[java] JAVA

http://javasoft.com

[pjae] PJAE Personal Java Applcation Environment 1.1.1, Sun Microsystems
http://javasoft.com/products/personaljava

[j2me] Java 2 Micro Edition
http://java.sun.com/j2me/

[wjava] Wireless Java

http://wireless.java.sun.com/

[awt] Java AWT (Abstract Window Toolkit)
http://java.sun.com/products/jdk/awt/

[kvm] K Virtual Machine (K steht für *Kilo*byte)
http://java.sun.com/products/cldc/ds/

[jphone] JavaPhone API
http://java.sun.com/products/javaphone/

[cldc] CLDC (Connected Limited Device Configuration)
http://java.sun.com/products/cldc

[cldcr] Referenzimplementation von Sun Microsystems für j2me

http://www.sun.com/software/communitysource/j2me/cldc/download.html

[midp] MIDP (Mobile Information Device Profile) JSR-37

http://java.sun.com/products/midp/

[csl] Community Source Licensing
http://www.sun.com/software/communitysource/

[http] Hypertext Transfer Protocol - HTTP1.1, IETF document RFC2616
http://www.w3.org/Protocols/rfc2616/rfc2626

[3gpp] 3rd Generation Partnership Project
http://www.3gpp.org

[wapf] Wapforum - Gremium, welches die Standards für die WAP implementierung vorgibt
http://www.wapforum.org

[wap] WAP (Wireless Application Protocol)

http://www.wapforum.org

[wml] WML (Wireless Markup Language)

http://www1.wapforum.org/tech/terms.asp?doc=WAP-238-WML-20010626-p.pdf

[sym] Symbian

http://www.symbian.com

[mot] Motorola

http://www.motorola.de/mobiltel/public/produkte/datenblaetter/accompli008/datenblatt.shtml

[nokia] Nokia

http://www.nokia.com

[fnok] Forum Nokia

http://www.forum.nokia.com

[blue] "Balzer" Technologie für Bluemark

http://www.bluelark.com/

[brow] Informationen zu Browse-It

http://www.intellisync.com/s2_browse-it.shtml

[eudo] EudoaWeb von Quallcomm

http://www.qualcomm.com

[borl] Borland

http://www.borland.com

[jbuil] JBuilder - Borland

http://www.borland.com/jbuilder/

[jbmob] JBuilder Mobile Set

http://www.borland.com/jbuilder/mobileset

[dataq] Dataquest

http://www.dataquest.com

[mardev] Momentan auf dem Markt erhältliche Geräte für j2me

http://www.microjava.com/devices

[comp] Compaq

http://www.compaq.com

[micro] Microsoft

http://www.mircosoft.com

[palm] Palm Inc.

http://www.palm.com

[palmab] Palm – über die Firma

http://www.palm.com/de/about/

[palmsto] Firmengeschichte der Firma Palm Inc.

http://www.berghoff.net/palm/anffaq.htm

[palmmob] Palm Mobile Internet Kit – Web Clipping

http://www.palmtopmagazin.de/produkte/palmos/palm_software/buero/mobile_internet_kit.php3

[palmcw] CodeWarrior für den Palm

http://www.metrowerks.com/products/palm/

[psion] Psion PLC

http://www.psion.com

[http] http Protokollspezifikationen

http://www.w3.org/Protocols/

[siemens] Siemens

http://www.siemens.com

[motorola] Motorola

http://www.motorola.com

[cldcmar] Die momentan auf dem Markt verfügbaren Java-kompatiblen Mobiltelefone

http://www.microjava.com/devices

Anhang

Anhang A - Für MExE Classmark 2 relevante APIs

java.applet
java.awt
java.awt.datatransfer
java.awt.event
java.awt.image
java.awt.peer
java.beans
java.io
java.lang
java.lang.reflect
java.math
java.net
java.rmi
java.rmi.dgc
java.rmi.registry
java.rmi.server
java.security
java.security.acl
java.security.cert
java.security.interfaces
java.security.spec
java.sql
java.text
java.text.resources
java.util
java.util.jar
java.util.zip

PJAE-spezifische APIs

Double Buffering

GUI-Komponenten bei Geräten ohne Maus

Timer API

Anhang B - Auswahl an Software für Nokia Communicator 9210

Für den Nokia Communicator 9210 existieren Anwendungen von Drittherstellern, welche allerdings **nicht mit MExE** realisiert sind. Das zeigt, dass MExE bisher keine Bedeutung unter Anwendungsentwicklern erlangen konnte. Hier eine Auswahl:

Accenture

- Mobile Business-to-Consumer Anwendungen
- Business-to-Employee Anwendungen
- Zugriff auf Unternehmensinterne Daten
- Wireless Personal Wealth Management Application zur Erweiterung der Finanzdienstleistungen von Banken

Computer Associates (CA)

- vereinfacht die Handhabung, Verwaltung und Verteilung von heterogenen IT-Umgebungen

Celesta

- Minimierung der Airtime für Offline- und Online-Modus.

Citrix

- Zugriff auf Office-PCs mit dem Nokia 9210 Communicator

Flander

- Flander-passende VGA-Adapter

Lotus

- WAP-Zugriff auf E-Mail und PIM (Personal Information Management).

Oracle

- Datensynchronisation zwischen einer Oracle 8iLite Datenbank

REAL Networks

- Multimedia-Inhalte
- Video- und Musikformate wiedergeben.

SAP

- Zugriff auf SAP-Anwendungen
- Anbindungen für mobiles Datenbankmanagement.

Sybase

- Client-Software für den mobiler Zugriff auf Sybase Datenbanken

Tivoli

- Management Software

Geräte update

Neuinstallation von Software wenn der Mitarbeiter mit dem Gerät unterwegs ist

TOMTOM

Reiseplanung und Buchung

Stadt- und Routenpläne

Übersetzungsdienste

Spiele

Yellow

Übersetzer Pro'Lingua

Mobile Auftragserfassung

Kundendienst

Mitarbeitersteuerung

GPS Ortung (Global Positioning System)

Anhang C - myTimeWAP Quellcode

myTimeWAP.wml

```
<?xml version ="1.0" ?>
<!DOCTYPE wml PUBLIC "-//WAPFORUM//DTD WML 1.3//EN" "http://www.wapforum.org/DTD/wml13.dtd" >
<wml>
        <card id="myTime1" title ="myTime" newcontext ="true" >
        <p>
                Startzeit: <input title ="Startzeit" name ="startzeit" type="text" value ="00:00:00" /><br/>
                Endzeit: <input title="Endzeit" name="endzeit" type="text" value ="00:00:00" /><br/>
                Datum: <input title ="Datum" name="datum" type="text" value ="00.00.2000" /><br/>
                Kundennummer: <input title ="Kundennummer" name ="kundennummer" type="text" value =""/><br/>
                <!-- Die Ziel-URL kann mit WAP (wie in HTML) nicht vom Benutzer geändert werden -->
                Senden An: http://bachelorarbeit.gschreiber.com/myTime/empfang.php
        </p>

        <do type ="accept" label ="Daten Schicken">
                <go href="http://bachelorarbeit.gschreiber.com/myTime/empfang.php" method ="post" >
                        <postfield name="startzeit" value="$(startzeit)" />
                        <postfield name="endzeit" value ="$(endzeit)" />
                        <postfield name="kundennummer" value="$(kundennummer)" />
                        <postfield name="datum" value ="$(datum)" />
                </go>
        </do>
        </card >
</wml>
```

Anhang D - myTime Quellcode

mytime.MainClass.java

```
/**
 * Dies ist die Hauptklasse von myTime.
 */

import java.awt.*;
import com.symbian.epoc.awt.*;
import com.symbian.devnet.crystal.awt.*;
import java.util.Calendar;
import java.lang.Integer;
import javax.net.datagram.*;
import java.net.*;
import java.io.*;
import java.util.Vector;

/**
 * Die Klasse MainClass ist die Hauptklasse von myTime. Sie als erstes aufgerufen und
 * steuert alle weiteren Abläufe der Applikation.
 *
 * myTime ist ein mobiles Zeiterfassungssystem welches auf dem Nokia Communicator 9210
 * lauffähig ist.
 */

class MainClass extends CFrame implements CBAListener {

  public static HauptPanel mainPanel;
  public static ConfigPanel configPanel;

  TextField textField1 = new TextField("",20);
  TextField textField2 = new TextField("",20);
  TextField textField3 = new TextField("",20);
  TextField textField4 = new TextField("",20);

  Label label1 = new Label();
  Label label2 = new Label();
  Label label3 = new Label();
  Label label4 = new Label();

  public CBAHandler cba;

  final static int FIRST_BUTTON = EikCommandButtonGroup.BUTTON1;
  final static int SECOND_BUTTON = EikCommandButtonGroup.BUTTON2;
  final static int THIRD_BUTTON = EikCommandButtonGroup.BUTTON3;
  final static int FOURTH_BUTTON = EikCommandButtonGroup.BUTTON4;

  PopUp popUp = new PopUp(this, Version.TITLE, this);
  PopUp popUpVielText = new PopUp(this, Version.TITLE, this, true);

  AboutDialog aboutDialog;

  URLConnection connection;

public MainClass() {

        setTitle(Version.TITLE);

        mainPanel = new HauptPanel();
        configPanel = new ConfigPanel();

        com.symbian.epoc.awt.EikStatusPane.setStatusPaneStyle(EikStatusPane.WIDE);
        // fehler im System: muss verschoben werden, damit das Fenster richtig plaziert ist
        this.setLocation(92,0);
        this.resize(com.symbian.epoc.awt.CKONToolkit.getAvailableScreenRect().getSize());

        // kein eigenes Logo möglich

        init();

}

public static void main ( String args[] ) {
  new MainClass();
}

public void init() {

  cba = new CBAHandler (this);
  initMainScreen();

this.add(configPanel);
this.show();

this.add(mainPanel);
this.show();

configPanel.setVisible(false);
```

```
}

public void initMainScreen() {

  cba.setText(FIRST_BUTTON, "Startzeit\nfestlegen");
  cba.setText(SECOND_BUTTON, "Conifig");
  cba.setText(THIRD_BUTTON, "Zeit\nSenden");
  cba.setText(FOURTH_BUTTON, "Ende");
  cba.setDefaultButton(FIRST_BUTTON);
  cba.activate();

}

public void initConfigScreen() {

  Data.state = Data.CONFIG;

  cba.setText(FIRST_BUTTON, "OK");
  cba.setText(SECOND_BUTTON, "per\nSMS");
  cba.setText(THIRD_BUTTON, "per\nhttp");
  cba.setText(FOURTH_BUTTON, "");

  cba.setDefaultButton(FIRST_BUTTON);

}


public void sendSMS(String zielNummer, String nachricht) {
  try {

    //Nummer für Datagramm formatieren!
    zielNummer = "sms://"+zielNummer;

    Address zielAdresse = DatagramService.parseAddress(zielNummer);
    DatagramService service = DatagramService.getService(zielAdresse);
    service.send( new Datagram(nachricht.getBytes(), zielAdresse));
       popUp.showText("Folgende Daten wurden als SMS an",
              Data.getEmpfaenger(),
              "gesendet:",
              Data.getNachricht());
      popUp.setVisible(true);
     }
  catch( Exception e ) {
    popUpVielText.showMuchText("Fehler:\n"+e+"\nzielNummer:"+zielNummer+"\nNachricht:"+nachricht);
    //println( "Failed to send message: " + e );
    e.printStackTrace();
  }
}

public void sendHTTP (String httpNachricht, String httpZielAdresse) {
String nachrichtEncoded = URLEncoder.encode(httpNachricht);
try {
  URL url = new URL (httpZielAdresse);
    try {
      URLConnection connection = url.openConnection();
      connection.setDoOutput(true);
      java.io.PrintWriter out = new java.io.PrintWriter(connection.getOutputStream());
      // alle Daten aus dem Vektor an den Server schicken
      for ( int i=0; i<Data.getHTTPdataToSend().size(); i++) {
        out.println(Data.getHTTPdataToSend().elementAt(i));
      }
      out.close();

      BufferedReader in = new BufferedReader (new InputStreamReader(connection.getInputStream()));
      String feedback;
      String feedbackString = "";

      while ( (feedback = in.readLine() ) != null) {
        feedbackString = feedbackString + feedback + "\n";
      }
      in.close();

            popUpVielText.showMuchText("Folgende Daten werden per http an \n" +
            Data.getEmpfaenger() + "\n" +
            "gesendet:\n" +
            Data.getNachricht() + "\n" +
            "Antwort vom Server:\n" + feedbackString);
            popUpVielText.setVisible(true);
    }
    catch (IOException ioe) {
      popUpVielText.showMuchText("Fehler:"+ioe);
      popUpVielText.setVisible(true);
    }
}
catch (MalformedURLException e) {
  popUpVielText.println("Fehler:"+e);
  popUpVielText.setVisible(true);
}

}

public void cbaActionPerformed (CBAEvent e) {

  switch (e.getID()) {
```

```
  case FIRST_BUTTON:
    if (Data.state == Data.MAIN) {

      if (Data.startState == Data.STOP) {
        Data.startZeit = getJetztZeitString();
        Data.Datum = getJetztDatumString();
        popUp.setVisible(true);
        popUp.showText("Die Aktuelle Zeit wurde als Startzeit genommen:",
          Data.getStartZeit(), "Das aktuelle Datum ist:", Data.getDatum());

        mainPanel.updateDisplay();
        cba.setText(FIRST_BUTTON, "Endzeit\nfestlegen");
        Data.startState = Data.START;
        break;
      }

      if (Data.startState == Data.START) {

        Data.endZeit = getJetztZeitString();
        Data.Datum = getJetztDatumString();

        popUp.setVisible(true);
        popUp.showText("Die Aktuelle Zeit wurde als Endzeit genommen:",
          Data.getEndZeit(), "Das aktuelle Datum ist", Data.getDatum());

        mainPanel.updateDisplay();
        cba.setText(FIRST_BUTTON, "Startzeit\nfestlegen");
        Data.startState = Data.STOP;
        break;
      }

    }
    if (Data.state == Data.CONFIG) {
      initMainScreen();
      mainPanel.setVisible(true);
      mainPanel.updateDisplay();
      configPanel.updateData();
      Data.state = Data.MAIN;
      break;
    }
    break;
  case SECOND_BUTTON:
    if (Data.state == Data.MAIN){
      configPanel.updateData();
      initConfigScreen();
      Data.state = Data.CONFIG;
      mainPanel.setVisible(false);
      configPanel.setVisible(true);
      break;
    }
    if (Data.state == Data.CONFIG) {
      Data.sendenAls = Data.SMS;
      configPanel.updateDisplaySMSandHTTP();
      break;
    }
    break;

    case THIRD_BUTTON:
      if (Data.state == Data.MAIN) {
        if (Data.sendenAls == Data.SMS) {
          sendSMS(Data.getEmpfaenger(), Data.getNachricht());

          popUpVielText.showMuchText("Folgende Daten werden per SMS an \n" +
          Data.getEmpfaenger() + "\n" +
          "gesendet:\n " +
          Data.getNachricht());
          popUpVielText.setVisible(true);
        break;
       }
        if (Data.sendenAls == Data.HTTP) {
          sendHTTP(Data.getNachricht(), Data.getEmpfaenger());
        }
        break;
      }
      if (Data.state == Data.CONFIG) {
        Data.sendenAls = Data.HTTP;
        configPanel.updateDisplaySMSandHTTP();
        break;
      }
      break;

    case FOURTH_BUTTON:
    if (Data.state == Data.CONFIG) {
      initMainScreen();
      mainPanel.setVisible(true);

      mainPanel.updateDisplay();
      Data.state = Data.MAIN;
      break;
    }
    else {
      shutDown();
    }
      break;
}
```

```
}

          /**
           * aktuelle Zeit bestimmen und speichern
           */
  String getJetztZeitString() {
          Calendar calendarZeit = Calendar.getInstance();
          Integer zeitStundeInt = new Integer(calendarZeit.get(Calendar.HOUR_OF_DAY));
          Integer zeitMinuteInt = new Integer(calendarZeit.get(Calendar.MINUTE));
          Integer zeitSekundeInt = new Integer(calendarZeit.get(Calendar.SECOND));

          String zeitMinuteString = zeitMinuteInt.toString();
          if (zeitMinuteString.length() == 1) {
            zeitMinuteString = "0" + zeitMinuteString;
          }

          String zeitSekundeString = zeitSekundeInt.toString();
          if (zeitSekundeString.length() == 1) {
            zeitSekundeString = "0" + zeitSekundeString;
          }

          String aktuelleZeit = (zeitStundeInt.toString() + ":" +
            zeitMinuteString + ":" +
            zeitSekundeString);

            return aktuelleZeit;
}

          /**
           * aktuelles Datum bestimmen und als String zurückgeben
           */
  String getJetztDatumString() {
          Calendar calendarZeit = Calendar.getInstance();
          Integer datumTagInt = new Integer(calendarZeit.get(Calendar.DAY_OF_MONTH));
          Integer datumMonatInt = new Integer(calendarZeit.get(Calendar.MONTH));
          Integer datumJahrInt = new Integer(calendarZeit.get(Calendar.YEAR));

          String datumTagString = datumTagInt.toString();
          if (datumTagString.length() == 1) {
            datumTagString = "0" + datumTagString;
          }

          String datumMonatString = datumMonatInt.toString();
          if (datumMonatString.length() == 1) {
            datumMonatString = "0" + datumMonatString;
          }

          String aktuellesDatum = (datumTagString + "." +
            datumMonatString + "." +
            datumJahrInt);

            return aktuellesDatum;
}


}// Ende MainClass
```

mytime.PopUp.java

```
import java.awt.*;
import com.symbian.epoc.awt.*;
import com.symbian.devnet.crystal.awt.*;
import java.util.Calendar;
import java.lang.Integer;
import javax.net.datagram.*;
import java.net.*;


/**
 * Diese Klasse wird als PopUp von der Kalsse MainClass benutzt.
 */
class PopUp extends Dialog implements CBAListener
{

          /**
           * Breite und Höhe definieren.
           */
          final int WIDTH = 400;
          final int HEIGHT = 200;

          /**
           * Handler für die vier Knöpfe initialisieren.
           */
          private CBAHandler cba;

          final static int FIRST_BUTTON = EikCommandButtonGroup.BUTTON1;
          final static int SECOND_BUTTON = EikCommandButtonGroup.BUTTON2;
          final static int THIRD_BUTTON = EikCommandButtonGroup.BUTTON3;
          final static int FOURTH_BUTTON = EikCommandButtonGroup.BUTTON4;
```

```
  public static GridPanel panel = new GridPanel();
public static Panel textPanel = new GridPanel();

  Label label1 = new Label("                                                  ");
Label label2 = new Label("                                                 ");
  Label label3 = new Label("                                                  ");
Label label4 = new Label("                                                 ");

TextArea textArea = new TextArea(6,20);

MainClass mainClass;
Frame parent;

  /**
   * PopUp erzeugen mit Labeln zum Beschriften
   */
  PopUp( Frame parent, String title, MainClass mainClass)
  {
          super( parent , title , false );
       this.mainClass = mainClass;
       this.parent = parent;

          setSize( WIDTH, HEIGHT );
       setLocation(120,0);

          cba = new CBAHandler( this );
       cba.setText(FOURTH_BUTTON, "OK");
       cba.setDefaultButton(FOURTH_BUTTON);

       panel.add(label1,0,1);
       panel.add(label2,0,2);
       panel.add(label3,0,3);
       panel.add(label4,0,4);

       add(panel);

  }

  /**
   * PopUp erzeugen mit Textarea für viel Text
   */
  PopUp( Frame parent, String title, MainClass mainClass, boolean vielText)
  {
          super( parent , title , false );
       this.mainClass = mainClass;
       this.parent = parent;

      setSize( WIDTH, HEIGHT );
       setLocation(120,0);

          cba = new CBAHandler( this );
       cba.setText(FOURTH_BUTTON, "OK");
       cba.setDefaultButton(FOURTH_BUTTON);

       add(textArea);
  }

public void showText (String nachricht1, String nachricht2,
       String nachricht3, String nachricht4) {

  label1.setText(nachricht1);
  label2.setText(nachricht2);
  label3.setText(nachricht3);
  label4.setText(nachricht4);

  panel.setVisible(true);
}

public void showMuchText (String largeNachricht) {
textArea.setText(largeNachricht);
textPanel.setVisible(true);
}

public void print (String text) {
textArea.append(text);
textPanel.setVisible(true);
}

public void println (String text) {
  print ("\n"+text);
}

  /**
   * Den Handler implementieren und die Aktionen abfragen
   */
  public void cbaActionPerformed( CBAEvent e )
  {
          switch ( e.getID() )
          {
                  case FOURTH_BUTTON:
                           setVisible( false );
                           break;
          }
  }
```

```
        /**
         * setVisable Methode überschreiben
         */
        public void setVisible( boolean visible )
        {
                 if (visible) {
                cba.activate();
                 }
                 if (!visible) {
                cba.deactivate();
                 }
                 super.setVisible( visible );
        }
} // Ende PopUp
```

mytime.HauptPanel.java

```
import java.awt.*;
import java.awt.event.*;
/**
 * @version 0.05
 * Hauptpanel, dass anfangs angezeigt wird
 */

class HauptPanel extends com.symbian.devnet.crystal.awt.GridPanel
{

    final static int SMS = 0;
    final static int HTTP = 1;

  TextField textField1;
  TextField textField2;
  TextField textField2a;
  TextField textField4;

  Label label1;
  Label label2;
  Label label2a;
  Label label3;
  Label label31;
  Label label4;

    Font font = new Font ("SansSerif",0,10);

  //Pannel aufbauen
  public HauptPanel() {

  constraints.weightx = 1.0;
  constraints.weighty = 1.0;
  constraints.insets = new Insets( 1, 1, 1, 1 );

  Dimension textFeldgroesse = new Dimension(20,20);

  final int spalte1 = 0;
  final int spalte2 = 1;
  int zeile = 0;


  label1 = new Label();
  add(label1,spalte1,zeile);
  label1.setText("Startzeit:");

  textField1 = new TextField("",10);
  add(textField1,spalte2,zeile);
  textField1.setText(Data.getStartZeit());

  zeile ++;

  label2 = new Label();
  add(label2,spalte1,zeile);
  label2.setText("Endzeit:");

  textField2 = new TextField("",10);
  add(textField2,spalte2,zeile);
  textField2.setText(Data.getEndZeit());

  zeile ++;

  label2a = new Label();
  add(label2a,spalte1,zeile);
  label2a.setText("Datum:");

  textField2a = new TextField("",10);
  add(textField2a,spalte2,zeile);
  textField2a.setText(Data.getDatum());

  zeile ++;

  label3 = new Label();
  add(label3,spalte1,zeile);
```

```
  label3.setText("Senden als:");

  label31 = new Label(Data.sendenAls);
  add(label31,spalte2,zeile);

  zeile ++;

  label4 = new Label();
  add(label4,spalte1,zeile);
  label4.setText("An:");

  textField4 = new TextField(Data.getEmpfaenger(), 20);
  add(textField4,spalte2,zeile);

}

public void updateData () {
Data.startZeit = textField1.getText();
Data.endZeit = textField2.getText();
}

public void updateDisplay () {
textField1.setText(Data.getStartZeit());
textField2.setText(Data.getEndZeit());
textField2a.setText(Data.getDatum());
label31.setText(Data.sendenAls);
textField4.setText(Data.getEmpfaenger());
}


}
```

mytime.ConfigPanel.java

```
import java.awt.*;
import com.symbian.epoc.awt.*;
import com.symbian.devnet.crystal.awt.*;

class ConfigPanel extends com.symbian.devnet.crystal.awt.GridPanel
{

  TextField textField1;
  TextField textField1a;
  TextField textField2;
  TextField textField3;
  TextField textField4;
  TextField textField41;
  TextField textField5;

  Label label1;
  Label label1a;
  Label label2;
  Label label3;
  Label label4;
  Label label41;
  Label label5;
  Label label51;

  //Pannel aufbauen
  public ConfigPanel() {

  constraints.weightx = 1.0;
  constraints.weighty = 1.0;
  constraints.insets = new Insets( 2, 2, 2, 2 );

  Dimension textFeldgroesse = new Dimension(20,20);

  label1 = new Label();
  add(label1,0,0);
  label1.setText("Nachricht:");

  textField1 = new TextField("",30);
  add(textField1,1,0);
  textField1.setText(Data.nachricht);

  label1a = new Label();
  add(label1a,0,1);
  label1a.setText("Kundennummer:");

  textField1a = new TextField("",30);
  add(textField1a,1,1);
  textField1a.setText(Data.kundenNummer);


  label2 = new Label();
  add(label2,0,2);
  label2.setText("SMS-Nummer");

  textField2 = new TextField("",20);
  add(textField2,1,2);
  textField2.setText(Data.smsTelefonNummer);
```

```
  label3 = new Label();
  add(label3,0,3);
  label3.setText("URI:");

  textField3 = new TextField("",30);
  add(textField3,1,3);
  textField3.setText(Data.httpAdress);

  label4 = new Label();
  add(label4,0,4);
  label4.setText("Senden als:");

  label41 = new Label(Data.sendenAls);
  add(label41,1,4);

/**
  label5 = new Label();
  add(label5,0,5);
  label5.setText("An:");

  textField5 = new TextField(Data.getEmpfaenger(),30);
  add(textField5,1,5);
  textField5.setEditable(false);
**/
}

public void updateData () {
Data.nachricht = textField1.getText();
Data.kundenNummer = textField1a.getText();
Data.smsTelefonNummer = textField2.getText();
Data.httpAdress = textField3.getText();
Data.sendenAls = label41.getText();
}

public void updateDisplaySMSandHTTP () {
label41.setText(Data.sendenAls);
}

} //ende Conifg Klasse
```

mytime.Data.java

```
import java.net.URLEncoder;
import java.util.Vector;
/**
 * Title:        Data
 * Description:  Klasse für Daten
 * Copyright:    Copyright (c) 2002
 * Company:      gschreiber.com
 * @author:      Gabriel Schreiber
 * @version 0.02
 */

class Data {
        /**
         * Allgemeines
         */
        static String nachricht = "Zeiterfassung von myTime";
        static String kundenNummer = "Kundennummer";

        /**
         * Zu Allgemeines gehörige Zugriffsmethoden
         */
        public static String getNachricht() {
          String localNachricht ="";
          if (getStartZeit() != "" || getEndZeit() != "") {
            localNachricht = "Es wurde die Zeit ";
            if (getStartZeit() != "") {
              localNachricht = (localNachricht + "von " + getStartZeit() + " ");
            }
            if (getEndZeit() != "") {
              localNachricht = (localNachricht + "bis " + getEndZeit() + " ");
            }
            if (getDatum() != "") {
              localNachricht = (localNachricht + "am " + getDatum() + " ");
            }
            localNachricht = localNachricht + "aufgezeichnet";
          }
          if (getStartZeit() == "" && getEndZeit() == "") {
            localNachricht = "Es wurden keine Zeiten genommen ";
          }
          localNachricht = localNachricht + "-" + Data.nachricht;
          return localNachricht;
        }

        /**
         * Gibt die Kundennummer als String zurück
         */
        public static String getKundennummer() {
          return Data.kundenNummer;
```

```
        }

        /**
         * Sendedaten
         */
        static String httpAdress = "http://bachelorarbeit.gschreiber.com/myTime/empfang.php";
        static String smsTelefonNummer = "+491724012274";
        static String sendenAls = "http-Request";         //sms=Default - oder http im Config-Menue
        static final String SMS = "sms";
        static final String HTTP = "http-Request";

        /**
         * Zeiterfassungsdaten
         */
        static String startZeit = "noch keine Startzeit definiert";
        static String endZeit = "noch keine Endzeit definiert";
        static String Datum = "noch kein Datum definiert";
        /**
         * Zu Zeiterfassungsdaten gehörige Zugriffsmethoden
         */
        public static String getStartZeit() {
          if (startZeit == "") {
            return "noch keine Startzeit definiert";
          }
          if (startZeit != "") {
              if (startZeit == "noch keine Startzeit definiert") {
                return "";
              }
          }
        return startZeit;
        }

        public static String getEndZeit() {
          if (endZeit == "") {
            return "noch keine Endzeit definiert";
          }
          if (endZeit != "") {
            if (endZeit == "noch keine Endzeit definiert") {
              return "";
            }
          }
        return endZeit;
        }

        public static String getDatum() {
          if (Datum == "") {
            return "noch kein Datum definiert";
          }
          if (startZeit != "") {
              if (Datum == "noch kein Datum definiert") {
                return "";
              }
          }
        return Datum;
        }

        /**
         * Daten für Entwurfsmuster "State"
         */
        static final String MAIN = "main";
        static final String CONFIG = "config";
        static String state = MAIN;

        static final String START = "start";
        static final String STOP = "stop";
        static String startState = STOP;


/**
 * Zugriffsmethoden - abhängige Variablen
 */
public static String getEmpfaenger() {
  if (sendenAls == SMS){
    return (smsTelefonNummer);
  }
  if (sendenAls == HTTP) {
    return (httpAdress);
  }
  //zur Sicherheit Default zurückliefern
  return smsTelefonNummer;
}

public static void setEmpfaenger(String empfaenger) {
  if (sendenAls == SMS) {
  smsTelefonNummer = empfaenger;
  }
  if (sendenAls == HTTP) {
  httpAdress = empfaenger;
  }
}

public static Vector getHTTPdataToSend () {
  Vector zuUebertragendeDaten = new Vector();

  zuUebertragendeDaten.addElement("nachricht=" + URLEncoder.encode(Data.getNachricht()));
```

```
  zuUebertragendeDaten.addElement("startZeit=" + URLEncoder.encode(Data.getStartZeit()));
  zuUebertragendeDaten.addElement("endZeit=" + URLEncoder.encode(Data.getEndZeit()));
  zuUebertragendeDaten.addElement("kundennummer=" + URLEncoder.encode(Data.getKundennummer()));
  zuUebertragendeDaten.addElement("datum=" + URLEncoder.encode(Data.getDatum()));
  zuUebertragendeDaten.addElement("empfaenger=" + URLEncoder.encode(Data.getEmpfaenger()));

  return zuUebertragendeDaten;
}

} // ende Klasse Data
```

mytime.Version.java

```
/**
*
* @author Gabriel Schreiber
* @version 0.05, 30.3.2002
*/

class Version {
  static final String TITLE         = "myTime";
  static final String DESCRIPTION_1 = "Java Programm um Zeit zu nehmen";
  static final String DESCRIPTION_2 = "und zu versenden";
  static final String VERSION       = "0.75";
  static final String DATE          = "26.03.2002";
  static final String COPYRIGHT     = "(c) gschreiber.com";
  static final String AUTHOR        = "Gabriel Schreiber";

  static final String[] DIALOG_TEXT =
         {
               DESCRIPTION_1,
                  DESCRIPTION_2,
                  "Version " + VERSION + " - " + DATE,
                  COPYRIGHT,
                  AUTHOR
         };
}
```

Anhang E - myTimeMIDP Quellcode

mytiemmidp.myTimeMIDlet.java

```
package mytimemidp;

import javax.microedition.midlet.MIDlet;
import javax.microedition.lcdui.*;
import javax.microedition.io.*;
import java.util.Calendar;

import java.io.InputStream;
import java.io.InputStreamReader;
import java.io.IOException;

public class myTimeMIDlet extends MIDlet implements CommandListener {

  private final static String startZeit = "Startzeit";
  private final static String endZeit = "Endzeit";
  private final static String zeitSenden = "Zeit Senden";
  private final static String sendenAn = "Senden An...";
  private final static String kundennummer = "Kundennummer";

  private Command endCommand;
  private Command zurueckCommand;
  private Command okCommand;
  private Display display;
  private TextBox textBoxURL;
  private TextBox textBoxKundennemmer;

  public myTimeMIDlet() {
    display = Display.getDisplay(this);

    // Commands erzeugen
    endCommand = new Command("Ende", Command.EXIT, 1);
    zurueckCommand = new Command("Zurück", Command.BACK, 1);
    okCommand = new Command ("OK", Command.OK, 1);
  }

  // das Menue erzeugen
  public void startApp() {
    // Eine Liste erzeugen und auf IMPLICIT setzen
    List list = new List("myTimeMIDlet", Choice.IMPLICIT);

    // Die Auswahlmöglichkeiten hinzufügen
    list.append(startZeit, null);
    list.append(endZeit, null);
    list.append(zeitSenden, null);
    list.append(sendenAn, null);
    list.append(kundennummer, null);

    // nur den ENDE Knopf konfigurieren
    list.addCommand(endCommand);
    list.setCommandListener(this);

    display.setCurrent(list);
  }

  // nicht benötigt, muss aber in einem MIDlet vorhanden sein
  public void pauseApp() {}

  // nicht benötigt, muss aber in einem MIDlet vorhanden sein
  public void destroyApp(boolean unconditional) {}

  // Startzeit gesetzt als Alert für 2 Sekunden anziegen
  public void startZeitGesetzt() {
    Alert startZeitAlert = new Alert("Startzeit:");
    startZeitAlert.setTimeout(2000);
    startZeitAlert.setType(AlertType.WARNING);

    // Daten erfassen
    Data.startZeit = getJetztZeitString();
    Data.Datum = getJetztDatumString();

    startZeitAlert.setString("Startzeit wurde auf " + Data.startZeit + " gesetzt");

    display.setCurrent(startZeitAlert);
  }

  // Endzeit gesetzt als Alert für 2 Sekunden anziegen
  public void endZeitGesetzt() {
    Alert endZeitAlert = new Alert("Startzeit:");
    endZeitAlert.setTimeout(2000);
    endZeitAlert.setType(AlertType.WARNING);

      //Daten erfassen und speichern
      Data.endZeit = getJetztZeitString();
      Data.Datum = getJetztDatumString();

    endZeitAlert.setString("Endzeit wurde auf " + Data.startZeit + " gesetzt");
```

```
  display.setCurrent(endZeitAlert);
}

// eine beliebige nachricht für 2 Sekunden anziegen (für Feedback vom Server)
public void nachrichtAnzeigen(String nachricht) {
  Alert nachrichtAlert = new Alert("Nachricht:");
  nachrichtAlert.setTimeout(2000);
  nachrichtAlert.setType(AlertType.WARNING);
  nachrichtAlert.setString(nachricht);

  display.setCurrent(nachrichtAlert);
}

// einstellen, wohin die Zeit geschickt werden soll
public void configAnzeigen () {

  // TextField erzeugen
  textBoxURL = new TextBox("Senden an:",
    Data.httpAdress, 256, TextField.URL);

  // Komandos und Listener hinzufügen
  textBoxURL.addCommand(okCommand);
  textBoxURL.setCommandListener(this);

  display.setCurrent(textBoxURL);
}

// Kudennummer eingeben
public void kundennummerEingeben () {

  // TextField erzeugen
  textBoxKundennemmer = new TextBox("Kundennummer:",
              Data.kundenNummer, 256, TextField.ANY);

  // Komandos und Listener hinzufügen
  textBoxKundennemmer.addCommand(okCommand);
  textBoxKundennemmer.setCommandListener(this);

  display.setCurrent(textBoxKundennemmer);
}


/**
 * Listener erzeugen für das gesamte MIDlet. Es wird jeweils abgefragt,
 * welches Displayable angezeigt wird, oder was ausgewählt wurde, um die
 * entsprechende Aktion aus zu lösen
 */
public void commandAction(Command command, Displayable screen) {

  // Prüfen, ob das Event von einem Listobjekt kam
  if (command == List.SELECT_COMMAND) {

    // Es war von einem Listobjekt, also die Labels anschauen und heruasfinden,
    // welches der User ausgewählt hat.
    List list = (List) screen;
    String auswahl = list.getString(list.getSelectedIndex());

    if (auswahl.equals(startZeit)) {
      startZeitGesetzt();
    }
    if (auswahl.equals(endZeit)) {
      endZeitGesetzt();
    }
    if (auswahl.equals(zeitSenden)) {
      nachrichtAnzeigen(schreibURL(Data.getEmpfaenger()));
    }
    if (auswahl.equals(sendenAn)) {
      configAnzeigen();
    }
    if (auswahl.equals(kundennummer)) {
      kundennummerEingeben();
    }

  }

  int commandType = command.getCommandType();

  // Wenn Zurück gedrückt wird, zurück zum List-Menü
  if (commandType == Command.BACK) {
    startApp();
  }
  // Wenn Ende gedrückt wurde, Anwendung beenden
  if (commandType == Command.EXIT) {
    notifyDestroyed();
  }
  // Wenn OK gedrückt wurde, Daten speichern und...
  if (commandType == Command.OK) {
    // War eine die textBoxURL aktiv, dann String auslesen und speichern
    if (display.getCurrent() == textBoxURL) {
      Data.httpAdress = textBoxURL.getString();
    }
    // War eine die textBoxKundennummer aktiv, dann String auslesen und speichern
    if (display.getCurrent() == textBoxKundennemmer) {
      Data.kundenNummer = textBoxKundennemmer.getString();
```

```
      }
      //... an den Anfang gehen
      startApp();
    }
  }

  public static String schreibURL (String url) {
  // URL zusammenbauen
  url = url + Data.getHTTPdataToSend();
  String feedback = null;
    try {
      boolean status = false;
      HttpConnection httpConnection = null;
      InputStream inputStream = null;

      httpConnection = (HttpConnection)Connector.open(url);
      httpConnection.setRequestMethod(HttpConnection.GET);

      try {
        // dir URL mit den Daten an den server senden
        // hier kann nicht die Klasse URL benutzt werden - diese existiert im
        // MIDP nicht. Statdessen kann die Klasse HttpConnection, eine
        // abgespeckte Variante verwendet werden.
        if (httpConnection.getResponseCode() == HttpConnection.HTTP_OK) {
          inputStream = httpConnection.openInputStream();
          InputStreamReader inputStreamReader = new InputStreamReader(inputStream);
          int laenge = (int) httpConnection.getLength();
          if (laenge > 0) {
            char[] chars = new char[laenge];
            int zeichen = inputStreamReader.read(chars);
            feedback = new String(chars);
          } else {
            StringBuffer stringBuffer = new StringBuffer();
            int charNummer;
            while ((charNummer = inputStream.read()) != -1) {
              stringBuffer.append((char) charNummer);
            }
            feedback = stringBuffer.toString();
          }
        } else {
          feedback = "Fehler: Server gab " + httpConnection.getResponseCode() + " zurück";
        }
      } catch (IOException ioExeption) {
        feedback = "Fehler: IOException.";
      }
      finally {
        if (inputStream != null) {
          inputStream.close();
        }
        if (httpConnection != null) {
          httpConnection.close();
        }
      }
    } catch (IOException ioe) {
      feedback = "Fehler: Konnte nicht mit der angegebenen URL verbinden";
    }
    return feedback;
  }

/**
* aktuelles Datum bestimmen und als String zurückgeben
*/
  String getJetztDatumString() {
    Calendar calendarZeit = Calendar.getInstance();
    Integer datumTagInt = new Integer(calendarZeit.get(Calendar.DAY_OF_MONTH));
    Integer datumMonatInt = new Integer(calendarZeit.get(Calendar.MONTH));
    Integer datumJahrInt = new Integer(calendarZeit.get(Calendar.YEAR));

    String datumTagString = datumTagInt.toString();
    if (datumTagString.length() == 1) {
      datumTagString = "0" + datumTagString;
    }

    String datumMonatString = datumMonatInt.toString();
    if (datumMonatString.length() == 1) {
      datumMonatString = "0" + datumMonatString;
    }
    String aktuellesDatum = (datumTagString + "." +
      datumMonatString + "." +
      datumJahrInt);

    return aktuellesDatum;
}

/**
* aktuelle Zeit bestimmen und speichern
*/
  String getJetztZeitString() {
    Calendar calendarZeit = Calendar.getInstance();
    Integer zeitStundeInt = new Integer(calendarZeit.get(Calendar.HOUR_OF_DAY));
    Integer zeitMinuteInt = new Integer(calendarZeit.get(Calendar.MINUTE));
    Integer zeitSekundeInt = new Integer(calendarZeit.get(Calendar.SECOND));

    String zeitMinuteString = zeitMinuteInt.toString();
    if (zeitMinuteString.length() == 1) {
```

```
      zeitMinuteString = "0" + zeitMinuteString;
    }

    String zeitSekundeString = zeitSekundeInt.toString();
    if (zeitSekundeString.length() == 1) {
      zeitSekundeString = "0" + zeitSekundeString;
    }

    String aktuelleZeit = (zeitStundeInt.toString() + ":" +
      zeitMinuteString + ":" +
      zeitSekundeString);

    return aktuelleZeit;
  }
} // ende myTimeMIDlet Klasse
```

mytimemidp.Data.java

```
package mytimemidp;

public class Data {
        /**
         * Allgemeines
         */
        static String nachricht = "Zeiterfassung von myTime";
        static String kundenNummer = "";

        /**
         * Zu Allgemeines gehörige Zugriffsmethoden
         */
        public static String getNachricht() {
          String localNachricht ="";
          if (getStartZeit() != "" || getEndZeit() != "") {
            localNachricht = "Es wurde die Zeit ";
            if (getStartZeit() != "") {
              localNachricht = (localNachricht + "von " + getStartZeit() + " ");
            }
            if (getEndZeit() != "") {
              localNachricht = (localNachricht + "bis " + getEndZeit() + " ");
            }
            if (getDatum() != "") {
              localNachricht = (localNachricht + "am " + getDatum() + " ");
            }
            localNachricht = localNachricht + "aufgezeichnet";
          }
          if (getStartZeit() == "" && getEndZeit() == "") {
            localNachricht = "Es wurden keine Zeiten genommen ";
          }
          localNachricht = localNachricht + "-" + Data.nachricht;
          return localNachricht;
        }

        /**
         * Gibt die Kundennummer als String zurück
         */
        public static String getKundennummer() {
          return Data.kundenNummer;
        }

        /**
         * Sendedaten
         */
        static String httpAdress = "http://bachelorarbeit.gschreiber.com/myTime/empfangMIDlet.php";
        static final String HTTP = "http-Request";

        /**
         * Zeiterfassungsdaten
         */
        static String startZeit = "noch keine Startzeit definiert";
        static String endZeit = "noch keine Endzeit definiert";
        static String Datum = "noch kein Datum definiert";
        /**
         * Zu Zeiterfassungsdaten gehörige Zugriffsmethoden
         */
        public static String getStartZeit() {
          if (startZeit == "") {
            return "noch keine Startzeit definiert";
          }
          if (startZeit != "") {
              if (startZeit == "noch keine Startzeit definiert") {
                return "";
              }
          }
        return startZeit;
        }

        public static String getEndZeit() {
          if (endZeit == "") {
            return "noch keine Endzeit definiert";
          }
          if (endZeit != "") {
```

```
            if (endZeit == "noch keine Endzeit definiert") {
              return "";
            }
          }
        return endZeit;
        }

        public static String getDatum() {
          if (Datum == "") {
            return "noch kein Datum definiert";
          }
          if (startZeit != "") {
              if (Datum == "noch kein Datum definiert") {
                return "";
              }
          }
        return Datum;
        }

/**
 * Zugriffsmethoden - abhängige Variablen
 */
public static String getEmpfaenger() {
    return (httpAdress);
}

public static void setEmpfaenger(String empfaenger) {
  httpAdress = empfaenger;
}

public static String getHTTPdataToSend () {
  String zuUebertragendeDaten = new String("");

  zuUebertragendeDaten = zuUebertragendeDaten + "?nachricht=" + encode(Data.getNachricht() + "\n");
  zuUebertragendeDaten = zuUebertragendeDaten + "startZeit=" + encode(Data.getStartZeit() + "\n");
  zuUebertragendeDaten = zuUebertragendeDaten + "endZeit=" + encode(Data.getEndZeit() + "\n");
  zuUebertragendeDaten = zuUebertragendeDaten + "kundennummer=" + encode(Data.getKundennummer() + "\n");
  zuUebertragendeDaten = zuUebertragendeDaten + "datum=" + encode(Data.getDatum() + "\n");
  zuUebertragendeDaten = zuUebertragendeDaten + "empfaenger=" + encode(Data.getEmpfaenger() + "\n");

  return zuUebertragendeDaten;
}

/**
 * nachfolgender Encoder stammt von
 * http://www.w3.org/International/O-URL-code.html
 */

  final static String[] hex = {
    "%00", "%01", "%02", "%03", "%04", "%05", "%06", "%07",
    "%08", "%09", "%0a", "%0b", "%0c", "%0d", "%0e", "%0f",
    "%10", "%11", "%12", "%13", "%14", "%15", "%16", "%17",
    "%18", "%19", "%1a", "%1b", "%1c", "%1d", "%1e", "%1f",
    "%20", "%21", "%22", "%23", "%24", "%25", "%26", "%27",
    "%28", "%29", "%2a", "%2b", "%2c", "%2d", "%2e", "%2f",
    "%30", "%31", "%32", "%33", "%34", "%35", "%36", "%37",
    "%38", "%39", "%3a", "%3b", "%3c", "%3d", "%3e", "%3f",
    "%40", "%41", "%42", "%43", "%44", "%45", "%46", "%47",
    "%48", "%49", "%4a", "%4b", "%4c", "%4d", "%4e", "%4f",
    "%50", "%51", "%52", "%53", "%54", "%55", "%56", "%57",
    "%58", "%59", "%5a", "%5b", "%5c", "%5d", "%5e", "%5f",
    "%60", "%61", "%62", "%63", "%64", "%65", "%66", "%67",
    "%68", "%69", "%6a", "%6b", "%6c", "%6d", "%6e", "%6f",
    "%70", "%71", "%72", "%73", "%74", "%75", "%76", "%77",
    "%78", "%79", "%7a", "%7b", "%7c", "%7d", "%7e", "%7f",
    "%80", "%81", "%82", "%83", "%84", "%85", "%86", "%87",
    "%88", "%89", "%8a", "%8b", "%8c", "%8d", "%8e", "%8f",
    "%90", "%91", "%92", "%93", "%94", "%95", "%96", "%97",
    "%98", "%99", "%9a", "%9b", "%9c", "%9d", "%9e", "%9f",
    "%a0", "%a1", "%a2", "%a3", "%a4", "%a5", "%a6", "%a7",
    "%a8", "%a9", "%aa", "%ab", "%ac", "%ad", "%ae", "%af",
    "%b0", "%b1", "%b2", "%b3", "%b4", "%b5", "%b6", "%b7",
    "%b8", "%b9", "%ba", "%bb", "%bc", "%bd", "%be", "%bf",
    "%c0", "%c1", "%c2", "%c3", "%c4", "%c5", "%c6", "%c7",
    "%c8", "%c9", "%ca", "%cb", "%cc", "%cd", "%ce", "%cf",
    "%d0", "%d1", "%d2", "%d3", "%d4", "%d5", "%d6", "%d7",
    "%d8", "%d9", "%da", "%db", "%dc", "%dd", "%de", "%df",
    "%e0", "%e1", "%e2", "%e3", "%e4", "%e5", "%e6", "%e7",
    "%e8", "%e9", "%ea", "%eb", "%ec", "%ed", "%ee", "%ef",
    "%f0", "%f1", "%f2", "%f3", "%f4", "%f5", "%f6", "%f7",
    "%f8", "%f9", "%fa", "%fb", "%fc", "%fd", "%fe", "%ff"
  };

  /**
   * Kodiert einen String in das "x-www-form-urlencoded" Format.
   *
   * @param s String der Kodiert werden soll
   * @return Der kodierte String
   */
  public static String encode(String s)
  {
    StringBuffer sbuf = new StringBuffer();
    int len = s.length();
    for (int i = 0; i < len; i++) {
      int ch = s.charAt(i);
```

```
      if ('A' <= ch && ch <= 'Z') {             // 'A'..'Z'
          sbuf.append((char)ch);
      } else if ('a' <= ch && ch <= 'z') {      // 'a'..'z'
          sbuf.append((char)ch);
      } else if ('0' <= ch && ch <= '9') {      // '0'..'9'
          sbuf.append((char)ch);
      } else if (ch == ' ') {                   // Leerzeichen
          sbuf.append('+');
      } else if (ch == '-' || ch == '_'         // nicht belegt
          || ch == '.' || ch == '!'
          || ch == '~' || ch == '*'
          || ch == '\'' || ch == '('
          || ch == ')') {
          sbuf.append((char)ch);
      } else if (ch <= 0x007f) {                // andere ASCII Zeichen
          sbuf.append(hex[ch]);
      } else if (ch <= 0x07FF) {                // nicht-ASCII <= 0x7FF
          sbuf.append(hex[0xc0 | (ch >> 6)]);
          sbuf.append(hex[0x80 | (ch & 0x3F)]);
      } else {                                          // 0x7FF < ch <= 0xFFFF
          sbuf.append(hex[0xe0 | (ch >> 12)]);
          sbuf.append(hex[0x80 | ((ch >> 6) & 0x3F)]);
          sbuf.append(hex[0x80 | (ch & 0x3F)]);
      }
    }
    return sbuf.toString();
  }

} // ende Klasse Data
```

Anhang F – Quellcode der serverseitigen Datei empfang.php

```
<?
/*
* Diese Datei sendet Formulardaten an die hardcodierte Emailadresse, die
* im ersten Teil deklariert wird.
*
* Variablen werden dynamisch eingelesen
* Definierte Variablen:
* $subject := Betreff
* $from := Absender
*
* Version 0.9
* (c) Gabriel Schreiber
*/

$toMail = "myTime@gschreiber.com ";
if ($from) {
$header = "From: " . $from;
}
else {
$header = "From: <myTime@gschreiber.com> myTime-Mailer";
$from = "<myTime@gschreiber.com> myTime-Mailer";
}

if (!$subject) {
$subject = "Feedback von myTime";
}

if ($REQUEST_METHOD == "GET") {
 while (list($key, $val) = each ($HTTP_GET_VARS)) {
  if ($key != 'subject' && $key != 'from' && $key != 'Submit' && $key != 'submit') {
   $message = $message . $key . "=" . $val ."\n";
  }
 }
}

if ($REQUEST_METHOD == "POST") {
 while (list($key, $val) = each ($HTTP_POST_VARS)) {
  if ($key != 'subject' && $key != 'from' && $key != 'Submit' && $key != 'submit') {
   $message = $message . $key . "=" . $val ."\n";
  }
 }
}

mail ($toMail, $subject, $message, $header);

echo $message;

?>
```