INSTITUT FÜR INFORMATIK
DER LUDWIG–MAXIMILIANS–UNIVERSITÄT MÜNCHEN

**Fortgeschrittenenpraktikum**

# Aufbau eines WAP-Testbeds und Implementierung einer prototypischen Anwendung

Florian Kirstein

| | |
|---|---|
| Aufgabensteller: | Prof. Dr. Heinz-Gerd Hegering |
| Betreuer: | Rainer Hauck<br>Norbert Wienold<br>Stephen Heilbronner (DeTeSystem) |
| Abgabetermin: | 31. Juli 2001 |

# Inhaltsverzeichnis

# 1 Einleitung

Mobiles Internet und M-Commerce - die mobile Abwandlung des E-commerce - sind Schlagworte, die man inzwischen täglich irgendwo zu sehen oder hören bekommt. Das Bedürfnis auf zentral gespeicherte Daten wie Termine, Adressen, Telefonnummern oder auch E-Mail Korrespondenz jederzeit und überall zugreifen zu können nimmt ständig zu und die Mobilfunkindustrie investiert viel Geld um diese Bedürfnisse zu befriedigen. Erreicht werden soll die neue Mobilität durch WAP, dem Wireless Application Protocol, welches verspricht Internetinhalte direkt über das Mobiltelefon erreichbar zu machen.

## 1.1 Aufgabenstellung

Ziel dieser Arbeit soll es sein zu untersuchen, was WAP tatsächlich bietet und wie es technisch realisiert ist, um dann eine Test-Umgebung am Lehrstuhl für Netzwerk-Management aufzubauen, in der ein Prototyp für eine Intranet-Anwendung der DeTeSystem entwickelt wird.

Als Erstes soll dabei eine generelle Übersicht über die von der WAP Spezifikation definierten Verfahren erstellt werden, welche die technischen Hintergründe einer WAP-Anwendung näher erläutert und die nötige Infrastruktur vom Mobiltelefon bis zum Inhaltsanbieter beschreibt.

Anschließend erfolgt die Evaluation verschiedener Entwicklungswerkzeuge, die sich aus den Anforderungen der DeTeSystem ergeben, und dann im Testbed integriert werden sollen. Hierbei wird vor allem untersucht, in wiefern sich die verschiedenen zur Verfügung stehenden Technologien für die Umsetzung des Prototypes und einer eventuell folgenden realen Applikation eignen.

Mittels der sich aus dieser Evaluation als geeignet herausstellenden Werkzeuge wird im Anschluss der Prototyp einer Applikation erstellt, die mobile Zugriffe von Mitarbeitern auf Intranetdaten der DeTeSystem ermöglicht. Mit diesem Prototyp soll es möglich sein die Praxistauglichkeit eines solchen Systemes mit den vorhanden Endgeräten zu erproben und die generelle Benutzerführung einer solchen Anwendung zu demonstrieren.

## 1.2 Aufbau der Arbeit

In den ersten beiden Kapiteln wird auf die Hintergründe sowie technischen Grundlagen von WAP eingegangen. Danach folgt ein Überblick über die Möglichkeiten WAP-Inhalte anzubieten und über das Testbed das zu diesem Zweck aufgebaut wurde. Am Ende steht die Beschreibung der Applikation, die als Prototyp für interne Vorführungen bei der DeTeSystem konzipiert und realisiert wurde.

# 2 Inhalte der WAP Spezifikation

## 2.1 Grundlegendes

WAP wurde vom WAP Forum[1], dem Zusammenschluss der meisten namhaften Firmen aus dem Umfeld mobiler Kommunikation, als Standard verabschiedet. Dieser definiert sowohl den Datentransport vom Inhaltsanbieter bis zum Endgerät über die vorhandene Infrastruktur des Internets als auch die Beschreibungssprache für die Inhalte.

Insbesondere ist zu beachten, dass zwar das Internet für den Transport der Inhalte zum Einsatz kommt, WAP jedoch einen eigenen Standard darstellt und nicht direkt mit dem World Wide Web zu tun hat. Es ist also nicht möglich mit einem WAP-Handy direkt auf WWW-Inhalte zuzugreifen, sondern nur auf speziell dafür angelegte WAP-Seiten.

Als Endgeräte sind bei WAP speziell Mobiltelefone anvisiert, weshalb der Standard auch explizit auf die beschränkten Möglichkeiten dieser Geräte sowohl in der Anzeige als auch in der Speicherkapazität eingeht, und auch die geringeren Bandbreiten die heutzutage mobil verfügbar sind berücksichtigt.

## 2.2 Versionen

WAP befindet sich in stetiger Weiterentwicklung. Die erste 1998 veröffentlichte Version 1.0 wurde von Geräteherstellern und Netzanbietern kaum implementiert, es gibt zwar ein Siemens Mobiltelefon das WAP 1.0 unterstützt und das erste WAP-fähige Handy auf dem deutschen Markt war, aber mangels Unterstützung durch die Netzbetreiber nicht wirklich verwendet werden konnte. 1999 wurde dann WAP 1.1[2] veröffentlicht, der Standard den alle aktuellen WAP-Handys verwenden. WAP 1.2 ist inzwischen verabschiedet, allerdings gibt es kaum Endgeräte die es implementieren. Möglicherweise wird hier auch erst mit dem nächsten Versionssprung zur Version 2.0, der Mitte 2001 erwartet wird, eine neue Generation von WAP-Browsern auf die Handys finden. In der nächsten WAP-Version sollen vor allem bessere Verschlüsselungs- und Schlüsselmanagementfunktionen enthalten sein um sichere mobile Transaktionen zu ermöglichen.

## 2.3 WML

WML, die Wireless Markup Language[3], ist der für den Applikationsentwickler wichtigste Teil der WAP-Spezifikation. Ähnlich wie HTML ist WML eine Tag-basierte Markup Sprache, die allerdings in XML spezifiziert ist. Dies hat eine deutlich striktere Syntaxdefinition zur Folge, ermöglicht aber auch eine einfachere Validierung von Dokumenten.

Anders als bei HTML ist ein WML-Dokument nicht einfach eine Seite die vom Endgerät angezeigt wird, sondern es wird ein Deck definiert, das mehrere Cards enthält, von denen jeweils nur eine auf einmal betrachtet wird. Dies hat den Vorteil, dass nicht zwingend bei jedem Verfolgen eines Links eine neue Seite über das langsame GSM-Netz abgerufen werden muss, denn zu einer Indexseite können z.B. die wichtigsten weiterführenden Seiten gleich als weitere Cards mitgesendet werden. Da bei den relativ kleinen von WAP übertragenen Datenmengen nicht die geringe Bandbreite sondern vor allen die hohe Latenz für das Geschwindigkeitsempfinden relevant ist, kann dies die Anwendung einer Applikation erheblich beschleunigen.

WML enthält Pflichttags und optionale Tags, was den Effekt hat, das nicht alle WAP-Browser der verschiedenen Handys den genau gleichen Sprachumfang implementieren. Man muss bei der Anwendung optionaler WML Bestandteile also immer darauf achten, dass diese entweder auf allen gängigen Mobiltelefonen implementiert sind, oder die Anwendbarkeit der Applikation nicht zwingend an diesen Tags hängt.

Ein Beispiel für eine einfache WML-Seite, die ein Bild und einen Verweis auf eine weitere Seite enthält:

```
<?xml version="1.0"?>
<!DOCTYPE wml PUBLIC "-//WAPFORUM//DTD WML 1.1//EN"
"http://www.wapforum.org/DTD/wml_1.1.xml">
<wml>
<card id="Portal" title="DeTeSystem">
<do type="prev"><prev/></do>
<p>
<img src="detelogo.wbmp" alt="DeTeSystem"/><br/>
<a href="search.php3">Datenbanksuche</a><br/>
</p>
</card>
</wml>
```

## 2.4 WML-Script

WML-Script[4] ist, ähnlich wie JavaScript im World Wide Web, eine Scriptsprache, die auf dem Browser im Endgerät ausgeführt wird. Hierdurch lässt sich eine gewisse Interaktivität auch ohne Serverinteraktion erreichen, allerdings besteht hier erst recht die Problematik der Kompatibilität mit den verschiedenen WAP-Browsern. Eine typische Anwendung ist die Verarbeitung von Eingabedaten in

Formularen, um z.B. einige Eingabefehler bereits vor Absendung des Formulares abzufangen.

## 2.5 WTAI

WTAI ist das Wireless Telephony Application Interface[5]. Es definiert Aufrufe um aus WAP-Applikationen heraus Telefonfunktionen auszulösen, so ist es z.B. denkbar eine über einen WAP-Index gefundene Telefonnummer direkt anzurufen oder aus einer WAP Anwendung heraus eine SMS zu versenden.

WTAI Funktionen können auf zwei Wegen von WML heraus aufgerufen werden. Einmal als URI, der z.B. über einen Link aufgerufen wird, und zum anderen über WMLScript Aufrufe. Der Aufruf um einen Telefonanruf zu starten sähe dabei beispielsweise so aus:

```
URI:
<a href="wtai://wp/mc;123456">123456 Anrufen</a>

WMLScript:
var errorcode = wtaVoiceCall.setup(123456;1);
if(i>=0) {
  Browser.setVar("Msg","Anruf OK");
} else {
  Browser.setVar("Msg","Fehler bei Anruf");
}
Browser.go("displayMsg");
```

In der aktuell relevanten Version 1.1 ist dies noch recht eingeschränkt und nur auf wenigen Endgeräten einsetzbar, es ist jedoch abzusehen, dass dies in den kommenden Geräteversionen mehr Relevanz haben wird.

# 3 Komponenten einer WAP Umgebung

## 3.1 Überblick

Abbildung 1 zeigt die typischerweise in einem WAP-Szenario zum Einsatz kommenden Komponenten, sowie die wichtigsten Kommunikationsprotokolle zwischen diesen. Die Funktionen der verschiedenen Teile dieses Setups werden im weiteren Verlauf dieses Kapitels im Detail erklärt.

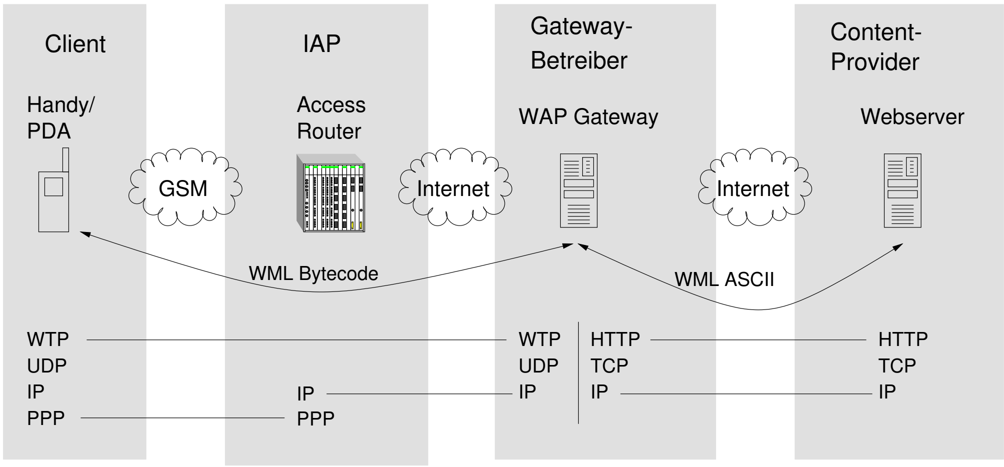


Abbildung 1: Die WAP Umgebung

## 3.2 Die Verbindung zum Mobiltelefon

Für den Anwender am interessantesten ist die Frage, auf welchem Weg er die Daten auf sein Endgerät bekommt. WAP spezifiziert dabei mehrere sogenannte Bearer Services, also Möglichkeiten die Daten zum Mobilteil zu übertragen. Die beiden gebräuchlichsten sind der Short Message Service SMS und - derzeit der einzig praxisrelevante Weg - Verbindungen über das Internet mittels IP über PPP. Letzteres wird in Mobilfunknetzen üblicherweise über eine normale Datenverbindung realisiert, inzwischen setzt sich jedoch ein neuer Standard, GPRS, der paketübermittelnd arbeitet, durch.

Unabhängig vom verwendeten Bearer Service wird dann der WML-Bytecode, eine komprimierte Fassung des WML-Decks, übermittelt. Die Wahl des Bearers ist für die WAP-Anwendung dabei transparent.

### 3.2.1 Der SMS Bearer

Der Vorteil einer Verbindung über SMS liegt auf der Hand: es werden nur Daten übertragen, wenn auch neue Information abgerufen wird. So lange man eine Seite liest oder mit der Zifferntastatur langwierig Zeichenfolgen eingibt fallen auch keine Kosten an. Problematisch ist hierbei jedoch die Abrechnung, zumindest in Deutschland wo der Short Message Service relativ teuer ist. Die meisten SMs, nämlich die mit den Antwortdaten auf eine Anfrage, werden vom Gatewaybetreiber zum Anwender geschickt, und somit müsste dieser dafür aufkommen. Die Übertragung eines kompletten WML-Decks mit bis zu 1397 Bytes wäre überdies sehr teuer, in Deutschland wird dieser Bearer daher von keinem Anbieter unterstützt.

### 3.2.2 Der CSD Bearer

CSD steht für Circuit Switched Data und bezeichnet eine Leitungsvermittelte Verbindung (im Gegensatz zu einer paketvermittelnden, zu der man den SMS Bearer zählen würde). Hier wird vom Mobiltelefon ein normaler Datenanruf aufgebaut, der i.d.R. im Festnetz auf einem Modem oder einem ISDN Router terminiert.

Auf der Luftschnittstelle des GSM Netzes sind die Daten dabei immer auf die gleiche Weise kodiert, erst beim GSM Netzbetreiber wird der Anruf je nach gewünschter Dienstekennung als ISDN Call nach V.110 bzw. V.120 (Asynchrone Datenverbindung mit Bitratenadaption) oder als analoger Datenanruf nach V.32 oder V.34 (übliche Modemstandards) aufgebaut. Die Datenrate ist dabei typischerweise 9600bps, es gibt jedoch auch einen Standard der durch Weglassen von Fehlerkorrekturinformationen 14400bps unterstützt und sich langsam auch bei den deutschen Netzbetreibern durchsetzt. Außerdem ist seit Ende 2000 bei E-Plus und D2 der neue Dienst HSCSD (High Speed Circuit Switched Data) verfügbar bei dem GSM-Seitig mehrere Kanäle zu je 9600 (oder 14400) bps zusammengeschaltet werden können. Allerdings gibt es erst wenige Mobiltelefone die auch WAP-Verbindungen über HSCSD unterstützen.

Über diese CSD Verbindung werden nun mittels PPP, dem Point-to-Point-Protocol, das auch im Festnetz üblich ist, IP-Pakete versendet und so eine Verbindung zum Internet hergestellt. Das Mobiltelefon hat also während der Sitzung eine normale IP-Adresse und empfängt und versendet IP Datenpakete. WAP spezifiziert dazu das verbindungslose Wireless Datagram Protocol WDP, das bei diesem Bearer auf UDP/IP aufsetzt.

Im Prinzip ist für den Zugang also jeder normale Internetprovider nutzbar, allerdings mit einer Einschränkung: die ISDN Protokolle V.110 und V.120 sind im Festnetz unüblich da dort keine Bitratenadaption nötig ist. Die meisten Provider bieten diese Protokolle daher nicht an. Eine Verbindung über die Modemprotokolle V.32 bzw. V.34 (welches jeweils zum Einsatz kommt hängt vom GSM Netzbetreiber ab) ist jedoch aus Providersicht genau das was auch ein Festnetzkunde verwendet, und damit überall verfügbar. Der Nachteil der Modemverbindung ist nicht der geringere Datendurchsatz, denn dieser wird zumindest ohne HSCSD ohnehin von der GSM Luftschnittstelle beschränkt, sondern der deutlich längere Verbindungsaufbau, welcher bereits zum kostenpflichtigen Gespräch gehört.

Alle Deutschen GSM-Provider bieten auch selbst Einwahlmöglichkeiten an, die mit i.d.R. 39Pf/Minute zwar billiger sind als ein mobiles Ferngespräch, aber teurer als der Anruf zu einem lokalen Provider mit einem der diversen Citytarife der Anbieter, so dass es sich für ausführlichere WAP-Sitzungen durchaus lohnt einen passenden Provider zu suchen.

### 3.2.3 Der GPRS Bearer

GPRS, der General Packet Radio Service, ist ein relativ neuer Mobilfunkstandard zur mobilen Datenübertragung, der wie der Name bereits andeutet paketvermittelnd arbeitet und außerdem deutlich höhere Datenraten als normale CSD Verbindungen erlaubt. Er basiert allerdings auf einem best-effort Prinzip, sind also viele Nutzer gleichzeitig aktiv hat jeder weniger Leistung zur Verfügung.

Über GPRS wird nun, wie auch schon über CSD, eine IP-Verbindung aufgebaut, wobei die IP-Pakete in GPRS Pakete verpackt werden. Die Einwahl erfolgt nicht mehr zu einem beliebigen Internetanbieter, sondern der Mobilfunkanbieter ist immer auch der IP-Provider, ansonsten funktioniert die Übertragung wie bereits beim CSD-Bearer dargestellt. Für das WAP-Gateway ist es also irrelevant, ob das Mobiltelefon seine Verbindung über CSD oder GPRS aufgebaut hat, es versendet die WDP Pakete in beiden Fällen über UDP/IP.

### 3.2.4 Bewertung

Der SMS Bearer ist der erste von realen Endgeräten implementierte Standard gewesen, hat sich jedoch bei den Netzbetreibern nie durchgesetzt und daher keine Praxisrelevanz. Der CSD/HSCSD Bearer ist heute Standard, hat jedoch den Nachteil recht hoher Verbindungskosten da die Datenverbindung auch weiterbesteht während Inhalte gelesen oder Formulare ausgefüllt werden. Abhilfe schafft hier der GPRS Bearer bei dem nur Daten übertragen werden wenn es nötig ist. Er wird nach Volumen abgerechnet und ist derzeit noch relativ teuer und nur von wenigen Endgeräten und Netzbetreibern unterstützt, wird aber mit Sicherheit der in Zukunft relevanteste Bearer Service sein und damit CSD im Bereich WAP ablösen.

## 3.3 Das WAP Gateway

Das WAP Gateway, oft auch als WAP Proxy bezeichnet, hat die Aufgabe den WML Bytecode aus dem Quelltext zu compilieren und über den jeweiligen Bearer zum Mobiltelefon zu übertragen. Hierzu kommt eine spezielle Software zum Einsatz, die auf einem Host läuft der per WDP vom Mobiltelefon aus erreichbar ist, also bei CSD/GPRS über die verwendete Einwahl per IP erreicht werden kann oder beim SMS-Bearer mit einem SMS-Center verbunden ist.

Bei den Netzbetreiberzugängen steht das WAP-Gateway direkt beim Betreiber, verwendet man einen anderen Provider stellt dieser im Allgemeinen kein eigenes Gateway zur Verfügung. Es gibt jedoch öffentlich verwendbare Gateways im Internet und es existiert auch eine freie Software, Kannel, mit der man sich ein eigened Gateway aufsetzen kann. Im Rahmen des Testbeds wird noch näher auf Kannel eingegangen.

Das WAP-Gateway bekommt nun auf verschiedenen Wegen WML Decks in ihrer ASCII Form und wandelt diese in den WML-Bytecode um, der dann per WDP an das Mobilteil versendet wird. Die übliche Quelle für diese WML Decks ist ein normaler Webserver, von dem das Gateway die WML-Seiten wie ein Webbrowser per HTTP abholt. Es gibt jedoch auch noch andere Möglichkeiten, so können z.B. über eine Filterapplikation auch normale HTML-Seiten in WML umgewandelt werden, dies ist aber prinzipbedingt eine extreme Informationsreduktion und die meisten Webseiten sind für eine derartige Umwandlung nicht geeignet.

# 4 Das WAP Testbed

## 4.1 WAP-SDKs

Als erste Komponente für das Testbed wurden die WAP SDKs von Nokia und Ericsson unter Windows installiert. Die SDKs bieten handyähnliche WAP-Browser auf dem PC, im Unterschied zu einem normalen Handy besitzen sie jedoch deutlich bessere Debuggingmöglichkeiten bei Fehlern im WML-Code. Außerdem benötigen sie kein WAP-Gateway, sondern wandeln den WML-Source selbst in Bytecode um.

Die Installation beider SDKs ist jeweils gut dokumentiert und gestaltet sich unter Windows NT problemlos, weshalb dies hier nicht weiter ausgeführt werden muss.

### 4.1.1 Das Nokia SDK

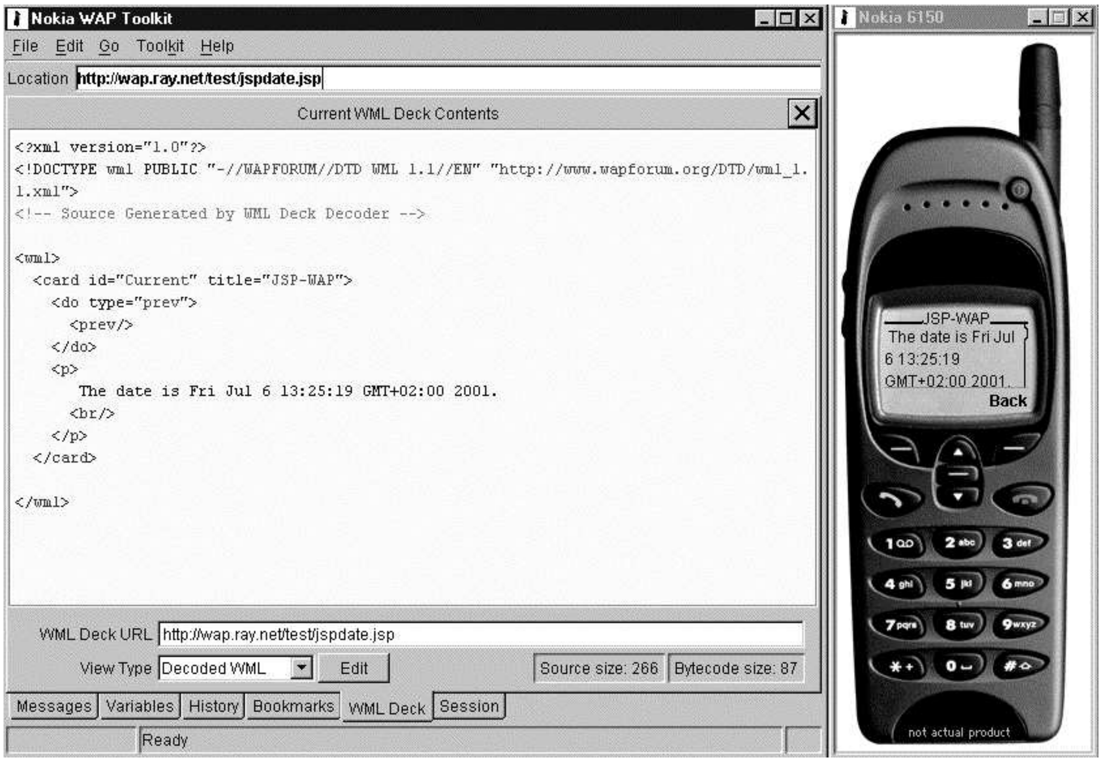


Abbildung 2: Das Nokia WAP SDK

Das Nokia SDK[6] besteht aus dem WAP-Browser, der in Form eines Handys dargestellt wird, und dem Arbeitsfenster in dem die diversen Zusatzfunktionen untergebracht sind. Außerdem liegt dem SDK eine WML und WML-Script Referenz bei die auch auf die Nokiaspezifischen Implementierungsdetails eingeht.

Das wichtigste Display zum Entwickeln ist dabei das auch in Abbildung 2 dargestellte WML-Deck, hier kann man den WML-Code und vor allem eventuelle Fehler darin sehen, sowie die Quell- und Bytecodegrössen.

Aber auch die anderen Displays erweisen sich beim Arbeiten als sehr nützlich, so ist es z.B. möglich, die bei WML-Script verwendeten Variablenwerte direkt zu beobachten und Informationen über alle von der aktuellen Session verwendeten Dateien zu bekommen...

### 4.1.2 Das Ericsson SDK

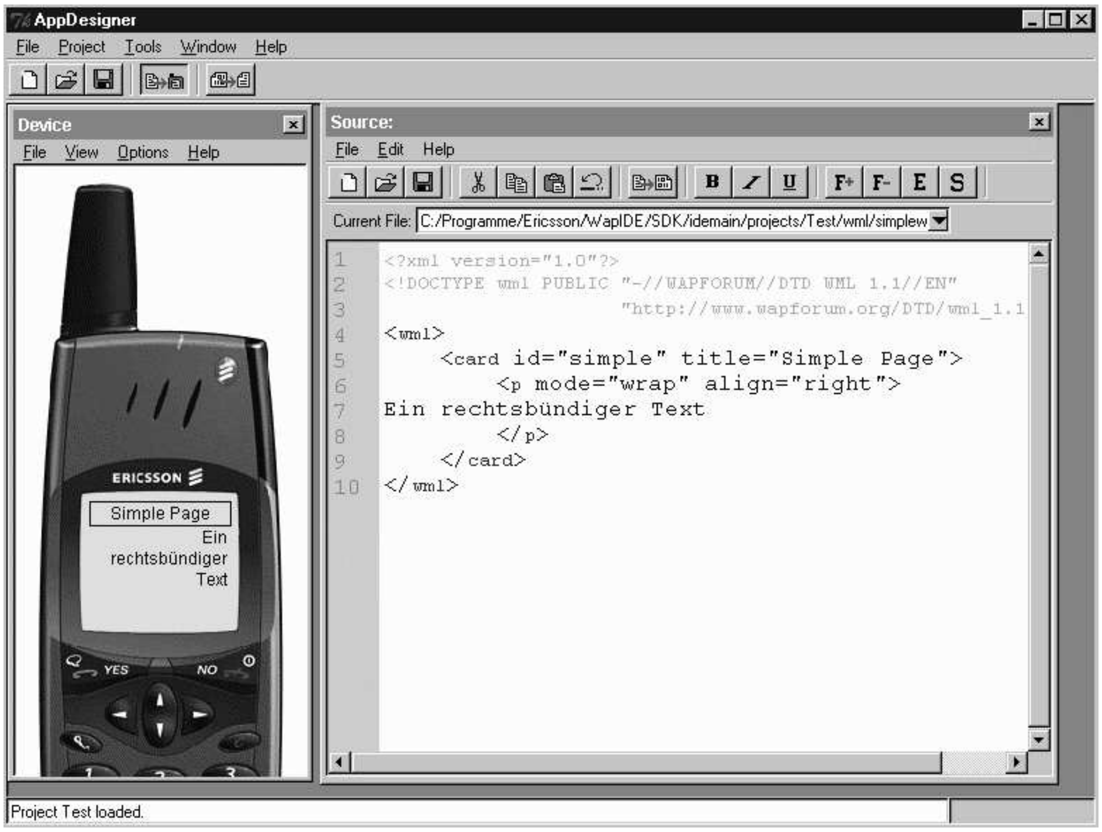


Abbildung 3: Das Ericsson WAP SDK

Das Konzept des Ericsson SDKs[7] geht in eine etwas andere Richtung, es enthält einen eigenen WML-Server der Dokumente ausliefert, sowie den in Abbildung 3 zu sehenden Editor, der beim Schreiben von WML-Seiten unterstützt und dazu auch eine Projektverwaltung bietet.

Für den Einstieg in WML ist es daher geeigneter als das Nokia SDK, es lassen sich die meisten WML-Elemente über Menüs erzeugen und der Anwender muss auch weniger selbst darauf achten valides XML zu erzeugen.

Will man jedoch eine dynamische WAP-Applikation auf einem eigenen Webserver entwickeln, ist das SDK weniger hilfreich als die Nokia Variante, da der Ericsson-Browser selbst kaum Debuggingmöglichkeiten hat. So fehlt alleine schon eine Sicht auf den Quelltext zur gerade dargestellten Seite, wenn diese nicht aus dem eigenen Editor sondern über das Netz kommt.

### 4.1.3 Grenzen der SDKs

Für die meisten Tests einer Applikation erwiesen sich beide SDKs als ausreichend. Es lohnt sich auch Seiten jeweils mit beiden zu betrachten da es doch einige Unterschiede gibt wie die beiden Hersteller die Benutzerführung bei einigen Elementen handhaben. Allerdings zeigten sich die SDKs gegenüber mancher Fehler im WML-Sourcecode toleranter als die Browser auf realen Endgeräten.

Insbesondere ein überschreiten des Grössenlimits für den Bytecode wird von den SDK-Browsern nicht besonders beachtet, das Nokia SDK zeigt aber zumindest die Bytecodegrösse permanent an, so dass man sie gut selber im Auge behalten kann - aber eben auch muss. Allgemein zeigte es sich, dass regelmäßige Tests mit einem echten Endgerät nicht verzichtbar sind, viele Fehler auf die man beim Browsen auf fremden WAP-Seiten trifft sind sehr wahrscheinlich darauf zurückzuführen, dass dies nicht ausreichend gemacht wird.

## 4.2 Webserver

Der Webserver ist die zentrale Komponente des Testbeds, von ihm aus werden die WML-Seiten an das WAP-Gateway ausgeliefert und auf ihm laufen auch die Applikationen die dynamische Seiten erzeugen.

Relevant für das WAP-Umfeld ist dabei, dass in der mime.types Datei spezielle Typen für WML-Decks und WBMP Grafiken angelegt werden müssen, dies geschieht durch die Einträge

```
text/vnd.wap.wml                wml
text/vnd.wap.wmlscript          wmls
image/vnd.wap.wbmp              wbmp
```

### 4.2.1 Betriebssystem und Software

Als Betriebssystem für den Webserver wurde Linux gewählt, aus Kompatibilitätsgründen mit der Testumgebung der DeTeSystem die SUSE Distribution. Darauf als eigentliche Webserversoftware der Apache[8] httpd, ergänzt um PHP[9], JServ und GnuJSP[11] für die dynamische Seitenerstellung. Diese Umgebung lässt sich jedoch auch problemlos auf anderen Linux-Distributionen oder anderen UNIX-Varianten einrichten, die verwendeten Konfigurationsdateien befinden sich in der Anlage.

## 4.3 Erzeugung dynamischer Seiten

Wie bereits erwähnt sind prinzipbedingt alle für Webseiten bekannten Verfahren zum erstellen dynamischer Seiten auch für WAP verwendbar. Die im Testbed implementierte Auswahl von Perl, PHP, Jaffa und JSP ergab sich aus den Wünschen der DeTeSystem. Neben der generellen Eignung für WAP wurde insbesondere auf die sich für die Erstellung des Prototypen aber auch eine eventuelle Weiterentwicklung ergebenden Kriterien geachtet, in zusammenarbeit mit der DeTeSystem wurden dabei herausgearbeitet:

- **Entwicklungszeit:** Um flexibel auf sich ergebende neue Anforderungen reagieren zu können ist es wichtig, dass mit dem gewählten Werkzeug schnell Ergebnisse erzielt und auch einfach verschiedene Variationen einer Lösung ausprobiert werden können.
- **Komplexität der Installation:** Damit der fertige Prototyp leicht in einer neuen Umgebung aufgesetzt und vorgeführt werden kann sollten die für den Betrieb nötigen Softwarekomponenten im Idealfall Bestandteil einer typischen Linuxinstallation sein, sich aber in jedem Fall mit geringem Zeitaufwand auf einem neuen Rechner einrichten lassen.
- **Java-Integration:** Da der Zugriff auf die Intranetdaten der DeTeSytem über ein komplexeres in Java implementiertes Framework erfolgt ist es wichtig, dass sich zumindestens die für den Prototyp entwickelten Ablaufkonzepte in eine Java-Umgebung übertragen lassen.
- **Sitzungsverwaltung:** Obwohl für den Prototypen nicht relevant, soll im Hinblick auf eine spätere Realisierung einer realen Applikation untersucht werden, wie gut sich mit dem jeweiligen Werkzeugen eine Sitzungsverwaltung realisieren lässt.

### 4.3.1 CGIs / Perl

Das Common Gateway Interface ist die älteste Methode zum dynamischen Ausgeben von Daten auf Webservern. Hierbei werden externe Programme auf dem Server aufgerufen und deren Ausgabe direkt an den Client gesendet. Parameter können dabei entweder im Environment (GET) oder auf der Standardeingabe (POST) übergeben werden.

Heutzutage ist Perl eine typische Sprache um CGI Programme zu implementieren, da sie sehr mächtige Möglichkeiten zur einfachen Datenverwaltung und Stringmanipulation bietet, und selbstverständlich Schnittstellen zum System und auch zu Datenbanken bietet. Ein weiterer Vorteil ist, dass CGI mit nahezu jeder Webserversoftware möglich ist und sich auch Perl, obwohl ursprünglich aus der Unix-Welt stammend, heutzutage ebenso problemlos unter Windows und anderen Systemen betreiben lässt.

Diese Kombination, PERL und CGI, erscheint damit ideal für Managementaufgaben, und für einfache Aufgaben lassen sich damit tatsächlich sehr schnell Erfolge erzielen, hier ein Beispiel für ein Script das die aktuelle Uptime und Systemlast eines Unix-Hosts anzeigt:

```
#!/usr/bin/perl
$|=1;
$uptime=`/usr/bin/uptime`; chop($uptime);
$host=`/bin/uname -n`; chop($host);
print "Content-Type: text/vnd.wap.wml\n\n";
print '<?xml version="1.0"?>
<!DOCTYPE wml PUBLIC "-//WAPFORUM//DTD WML 1.1//EN"
 "http://www.wapforum.org/DTD/wml_1.1.xml">
<wml>
<head>
<meta http-equiv="Cache-Control" content="no-cache"/>
</head>
<card id="uptime" title="Uptime">
<do type="prev"><prev/></do>
<p>
Uptime on Host ' . $host . '<br/>
' . $uptime . '<br/>
</p>
</card>
</wml>
';
```

Komplizierter wird es jedoch sobald nicht mehr einzelne unabhängige Anfragen gestellt werden, sondern eine Session aufgebaut werden soll z.B. um eine Authentifikation nur einmal durchführen zu müssen oder wenn Daten in mehreren Seiten nacheinander ausgegeben werden sollen, was aufgrund des WML-Deck Grössenlimits von 1400 Bytes oft nötig ist. Dies ist mit CGIs zwar möglich, man muss sich jedoch selbst um die ganzen Mechanismen wie die Vergabe und Speicherung einer Session ID kümmern.

Außerdem muss der gesamte WML Code durch print-Befehle oder ähnliches vom Script ausgegeben werden, was gerade für den nicht dynamischen Rahmencontent einer Seite unnötig umständlich ist.

### 4.3.2 PHP

Ähnlich zu Perl ist PHP[9] eine interpretierte Scriptsprache, die jedoch speziell für den Einsatz auf Webservern konzipiert wurde. Im Gegensatz zu CGIs, wo Programme ablaufen, die den kompletten Seitencode ausgeben, sind PHP-Programme in den Seitenquelltext integriert. Es werden also normale Seiten erstellt und an den dynamisch zu erzeugenden Stellen Codefragmente eingebaut die den entsprechenden Teil ausgeben.

Hier ein Beispielcode der das aktuelle Datum und die Uhrzeit ausgibt. Zu beachten ist, dass, da der Header bei PHP nicht vom Script sondern vom Webserver erzeugt wird, als allererstes mit dem Header Befehl der passende content-type gesetzt wird. Ebenfalls ist es zu empfehlen den xml-Kopf wie gezeigt als erstes per print auszugeben und nicht normal in den Seitenquelltext zu schreiben. Sonst passiert es leicht, dass dieser nicht ganz am Anfang der Seite steht, was zu einem der typischen Fehler führt die bei Tests mit einem Emulator nicht entdeckt werden, auf einem reales Endgerät jedoch einen Abbruch bewirken.

```
<?php
Header("Content-Type: text/vnd.wap.wml");
print "<?xml version=\"1.0\"?>\n";
?>
<!DOCTYPE wml PUBLIC "-//WAPFORUM//DTD WML 1.1//EN"
 "http://www.wapforum.org/DTD/wml_1.1.xml">
<wml>
<card id="Current" title="PHP3-WAP">
<do type="prev"><prev/></do>
<p>
<?php print(date( "l dS of F Y h:i:s A" )); ?> <br/>
</p>
</card>
</wml>
```

Die Interpretierung des Programmcodes wird bei PHP i.d.R. direkt durch ein im Webserver integriertes Modul erledigt, wodurch auch eine engere Integration mit dem Server möglich wird. Ein entscheidender Vorteil dieser Methode gegenüber CGIs ist es, dass das PHP Modul bei Datenbankanwendungen eine persistente Verbindung zur Datenbankengine aufrecht erhalten kann, während CGIs diese immer erst wieder neu aufbauen müssen, was Zeit kostet. Außerdem bringt PHP von sich aus ein Sessionmanagement mit das eine leichte Verwaltung sessionpersistenter Daten ermöglicht. Hierbei sind jedoch die Einschränkungen zu beachten, die am Ende dieses Kapitels zu Sessions und WAP allgemein genannt werden.

Alles in allem ist PHP damit sehr gut zum schnellen Entwickeln einfacher und auch komplexerer Anwendungen geeignet, und wurde deshalb nach einigen Tests auch als Sprache für den später beschriebenen Prototyp einer Intranet-Anbindung per WAP bei der DeTeSystem verwendet.

### 4.3.3 Java Servlets / Jaffa

Einen komplett anderen Ansatz verfolgt das von Ericsson angebotene Jaffa Toolkit, das speziell zur Entwicklung von Java Servlets für WAP-Applikationen konzipiert ist.

Servlets sind Java-Programme, die sich über das Servlet Interface an den Webserver koppeln und für diesen bei für sie bestimmten Anfragen den Content erzeugen. Da das Servlet permanent läuft ist eine Verwaltung Sessionpersistenter Daten hier auch ohne eine externe Datenbank möglich.

Das Jaffa Toolkit verfolgt zur Erstellung der Seiten einen objektorientierten Ansatz. Der Entwickler legt ein Objekt für das WML-Deck an und fügt diesem dann WML-Cards mit bestimmten Subelementen hinzu, und baut so eine logische Struktur der kompletten Seite im Speicher auf. Diese wird dann von durch Jaffa zur Verfügung gestellten Funktionen als WML generiert und ausgegeben.

Der Vorteil dieses eher komplexen Ansatzes ist es, dass das eigentliche Programm nicht direkt WML Tags erzeugen muss, und so auch eine gewisse Unabhängigkeit vom konkreten WML Standard erreicht wird. Ericsson wirbt damit, dass so z.B. ein Umstieg auf eine neuere WML Version ohne Anpassung der Applikation, einfach durch Update auf ein neueres Jaffa Toolkit möglich ist.

Da jedoch eine neuer WAP-Standard voraussichtlich vor allem neue Funktionalität bietet und nicht so sehr die in älteren Standards vorhandenen Tags komplett ersetzt, ist der reale Nutzen der sich für den Entwickler dadurch bietet je nach Anwendung nicht besonders relevant. Hinzu kommt, dass der Entwicklungszyklus einer Jaffa Applikation deutlich aufwendiger ist als bei den anderen hier vorgestellten Methoden, da bei jeder Änderung im Code jeweils die Klassen neu kompiliert und die laufenden Servlets am Webserver neu gestartet werden müssen.

Die Zielgruppe von Jaffa sind also eher Entwickler die eine komplexe Applikation mit vielen verschiedenen dynamischen Decks und Cards erstellen wollen, nicht jedoch um einfach und direkt mit verschiedenen WML-Strukturen zu experimentieren und kleinere Testanwendungen zu implementieren. Jaffa wurde im Testbed eingerichtet und mit den Beispielanwendungen getestet, da sich aber bereits dabei zeigte dass es für die angestrebten Anwendungen am wenigsten geeignet ist nicht weiter verwendet.

### 4.3.4 JSP

Da jedoch die DeTeSystem ihr Datenbank Framework in Java realisiert hat, sollte auch für den Fall einer Weiterentwicklung des Prototyps zu einer realen Applikation die Möglichkeit einer einfachen Integration evaluiert werden. Hierfür bot sich JSP[10] an, das einerseits ähnlich wie PHP eine in den Seitenquelltext integrierte Sprache ist, zum anderen jedoch auf Java basiert und zusätzlich zum in die Seiten integrierten Code noch sogenannte Beans unterstützt, eigenständige Javaklassen in die Seiten integriert werden können.

Auch hier ein Beispiel für eine WAP-Seite, wie bereits bei dem Beispiel in PHP wird das aktuelle Datum ausgegeben, wiederum ist zu beachten dass der xml-Kopf in der allerersten Zeile beginnt, der Umbruch davor ist hier nur aus drucktechnischen Gründen.

```
<%@ page contentType="text/vnd.wap.wml" %><%@ page session="false" %>
<?xml version="1.0"?>
<!DOCTYPE wml PUBLIC "-//WAPFORUM//DTD WML 1.1//EN"
"http://www.wapforum.org/DTD/wml_1.1.xml"><%@ page import="java.util.Date" %>
<wml>
<card id="Current" title="JSP-WAP">
<do type="prev"><prev/></do>
<p>
The date is <%= new Date() %>.
<br/>
</p>
</card>
</wml>
```

Da der Prototyp in PHP ausreichend war, galt es bei JSP vor allem zu testen, ob und wie gut es sich für WAP einsetzen lässt. Die beiden Probleme hierbei waren das Setzen des richtigen Content Headers, was sich durch ein einfaches Tag wie im Beispiel zu sehen am Anfang der Seite bewerkstelligen lässt, sowie die Sessionverwaltung die ein elementarer Bestandteil des Beans Konzeptes ist, da typische Beans persistent über eine Session erzeugt werden und so für Seitenübergreifende Datencontainer genutzt werden können.

Die Sessions werden von JSP mit Cookies im Browser abgelegt, daher müssen die im nächsten Kapitel erläuterten Besonderheiten bei WAP berücksichtigt werden.

Eine Beispielapplikation in JSP mit einer sitzungspersistenten Bean ist im Anhang zu finden, sie kann verwendet werden um einfach zu testen ob die Sitzungsverwaltung in einem konkreten Setup funktioniert.

### 4.3.5 Sessions und Cookies

Viele Anwendungen benötigen Daten die sich auf einen längeren Zeitraum als den Abruf einer einzelnen Seite beziehen. Um diese Daten zu speichern und vor allem von der Applikation aus dem User wieder zuzuordnen wird eine Sessionverwaltung benötigt. Hierfür haben sich im Web mehrere Techniken etabliert um Informationen beim Client zu halten, von denen sich aber nicht alle gleich gut auf WAP übertragen lassen.

Prinzipiell lassen sich zwei Arten unterscheiden, zum einen Verfahren bei denen die zu speichernden Daten selbst abgelegt werden, und solche bei denen die eigentlichen Daten serverseitig in einer Datenbank lagern und nur eine Kennung, die sogenannte Session-ID, auf dem Client gespeichert wird. Bei WAP kommt im Allgemeinen nur letzteres in Frage, da die Kapazitäten sinnvolle Datenmengen aufzunehmen auf den mobilen Endgeräten nicht gegeben sind.

Die erste Methode ist es, Daten in der URL zu kodieren. Hierbei wird die URL einer Seite mit Parametern angereichert, und diese auf der ja dynamisch erstellten Seite dann auch wieder an alle Verweise angehängt, wie WAP ist hier zu beachten dass dies ein Deck mit vielen Verweisen ziemlich aufblähen kann und man dann leicht an das Bytecodelimit stösst.

Einfacher für die Anwendungsprogrammierung sind Cookies, Informationsträger die z.B. im HTTP-Header übertragen, beim Client gespeichert und wieder abgerufen werden können. Während sich beim Web hier vor allem das Problem stellt, dass viele User die Cookies wegen Datenschutzbedenken vermeiden, sieht es bei WAP so aus, dass die wenigsten Endgeräte damit umgehen können. Dieses Problem umgehend gibt es jedoch WAP-Gateways, die das Speichern des Cookies für das mobile Endgerät übernehmen. Da die Verbindung zwischen Endgerät und WAP-Gateway statefull, also sitzungsbasiert ist, kann damit eine http-session in Cookies realisiert werden. Allerdings wird dies nicht von allen WAP-Gateways unterstützt, das von T-Mobil verwendet kann dies jedoch, so dass auch Anwendungen in JSP, die Sessiondaten benötigen, für die DeTeSystem kein Problem ergeben würden.

Als dritte Möglichkeit kommen im Web teilweise clientseitige Scripts zum Einsatz, die die Daten an geeigneten Stellen, z.B. einem immer sichtbaren Frame, ablegen. Dies lässt sich bei WAP, obwohl es mit WML-Script eine serverseitige Scriptsprache gibt, jedoch nicht sinnvoll realisieren und kommt daher nicht in Frage.

### 4.3.6 Vergleich

Die folgende Tabelle stellt die Vor- und Nachteile der einzelnen Systeme im Hinblick auf die für die DeTeSystem relevanten Merkmale noch einmal übersichtlich dar:

| | Perl | PHP | Jaffa | JSP |
|---|---|---|---|---|
| Entwicklungszeit | + | ++ | - | + |
| Installation | ++ | ++ | - | - |
| Java-Integration | − | − | + | ++ |
| Sizungsverwaltung | - | + | ++ | ++ |

## 4.4 Gateway

Um zum Testen mit realen Geräten nicht immer auf die teuren Einwahlports der Mobilfunkanbieter angewiesen zu sein, sollte im Testbed außerdem ein WAP-Gateway eingerichtet werden.

Zur Auswahl standen dabei eine Beta von Nokia für Windows sowie die freie Software Kannel für Unix, wobei letztere gerade erst begonnen hatte die nötige Funktionalität zu bieten, da die ersten Versionen vor allem auf SMS als Bearer ausgelegt waren. Die Entscheidung fiel dennoch auf Kannel, nicht zuletzt da der Windowsrechner im Testbed nur zum Testen mit den SDKs in Betrieb war, der Linuxrechner jedoch permanent laufen sollte.

Die Installation von Kannel gestaltete sich in der ersten getesteten Version noch etwas kompliziert aufgrund von Problemen mit der xml-Bibliothek, aktuelle Versionen lassen sich jedoch auf aktuellen Linux-Systemen problemlos durchcompilieren.

Kannel ist modular aufgebaut und sowohl auf SMS als auch auf IP als Transportmedium ausgelegt. Je nachdem welches man benötigt setzt man das entsprechende Bearer-System auf und konfiguriert das eigentliche WAP Gateway dafür. Die im Testbed verwendete Konfiguration für den IP Bearer und das WAP-Gateway findet sich in der Anlage.

# 5 Der Prototyp für die DeTeSystem

Ziel des Prototyps war, eine Applikation zu entwickeln, anhand derer bei der DeTeSystem interne Vorführungen einer WAP-Anbindung an Intranetdaten durchgeführt werden können, und dabei die bei einer solchen Anwendung auftretenden Probleme zu untersuchen.

## 5.1 Funktionalität

Die Applikation sollte drei Grundfunktionalitäten bieten, wobei der Schwerpunkt auf der ersten liegt: die Suche nach Mitarbeitern und ihren Daten wie Raum, Telefonnummer und Vertretung über eine Eingabemaske. Des weiteren sollte zumindest als Anschauungsobjekt eine Funktion zum Abrufen von e-Mail sowie ein einfacher Kantinenplan implementiert werden, dies wurde dann jedoch vom FoPra unabhängig durch Stephen Heilbronner von der DeTeSystem durchgeführt.

## 5.2 Implementierung

Da für den Protoyp in erster Linie relevant war, dass sich leicht und schnell Änderungen daran vornehmen lassen, um z.B. verschiedene Formularversionen und Deck-Aufbauten durchzutesten, wurde als Implementierungssprache PHP3 gewählt. Dies ist ausserdem im Gegensatz zu JServ/JSP auf den meisten Webservern bereits installiert oder zumindestens einfach als Paket zu installieren, so dass bei Bedarf eine schnelle Übertragung auf andere Server bei der DeTeSystem möglich ist.

Zur Datenablage wurde ein CSV-ASCII-File gewählt, in dem mittels in eine Pipe geöffnetem unix-grep gesucht wird. Gegebenenfalls liesse sich dies aber auch einfach durch einen Zugriff auf ein DBS ersetzen, da für die Tests mit dem Prototyp aber ohnehin nicht auf den realen Daten gearbeitet werden sollte war dies nicht nötig.

Vom Ablauf her wurde das gesamte Programm in eine PHP3 Datei abgelegt, die je nachdem ob Parameter vorhanden sind oder nicht eine leere Suchseite oder das Ergebnis ausgibt. Dadurch war es möglich den Code einigermassen simpel zu halten und z.B. den Code für die Suchmaske mehrfach zu verwenden, da jede Card die in einem Ausgabedeck vorkommen kann über eine Funktion erzeugt wird.

## 5.3 Userinterface

Das wichtigste Ziel bei der Benutzerführung einer WAP-Anwendung ist es, den Anwender mit möglichst wenigen Serveranfragen zu seinem Ziel zu bringen. Denn obwohl WAP den Handshake durch Verwendung von UDP statt TCP auf der Luftschnittstelle reduziert zeigte es sich, dass aufgrund der hohen Latenz einer mobilen Datenverbindung jeder Serverzugriff relativ lange dauert. Dies macht eine Anwendung bei vielen Zugriffen, auch wenn bei diesen jeweils nur wenig Daten übertragen werden, subjektiv recht langsam.

Ziel ist es also in einem WML-Deck schon möglichst viele Cards auf die der Anwender möglicherweise navigieren könnte mitzusenden. In der Anwendung wurde dies an mehrern Stellen realisiert, so enthält bereits die Einstiegsseite der Applikation die nötigen Cards für die Suchmaske, die erste Seite des Kantinenplanes und Einstiegsseite für die E-Mail Abfrage. Und bei einer Suche in der Mitarbeiterkartei werden, so es die Anzahl der Ergebnisse zulässt, bereits die Daten zu den ersten gefunden Einträgen mitgesendet. Bei einem eindeutigen Treffer wird zusätzlich zu den gefunden Daten gleich, so vorhanden, eine Card mit den Daten einer Vertretung, sowie die Suchmaske für eine weitere Suche mitgesendet.

Abbildung 4 zeigt den typischen Ablauf einer Suchanfrage mit nicht eindeutigem Ergebnis.


Abbildung 4: Ablauf einer Suchanfrage

# Literatur


[1] The Wap Forum,
*http://www.wapforum.org*

[2] Wapforum: Wap Specification Suite Version 1.1, 1999
*http://www.wapforum.org/what/technical_1_1.htm*

[3] Wapforum: Wireless Markup Language Specification Version 1.1, 1999
*http://www1.wapforum.org/tech/terms.asp?doc=SPEC-WML-19990616.pdf*

[4] Wapforum: WMLScript Language Specification Version 1.1, 1999
*http://www1.wapforum.org/tech/terms.asp?doc=SPEC-WMLScript-19990617.pdf*

[5] Wapforum: Wireless Telephony Application Interface Specification Version 1.1, 1999
*http://www1.wapforum.org/tech/terms.asp?doc=SPEC-WTAI-19990531.pdf*

[6] Nokia Mobile Phones: WAP Developer Toolkit
*http://forum.nokia.com/wapforum/main/1,6668,1_1_50,00.html*

[7] Ericsson WAP IDE
*http://www.ericsson.com/developerszone/*

[8] The Apache Webserver
*http://www.apache.org/*

[9] PHP Hypertext Processor
*http://www.php4.org/*

[10] SUN Java Server Pages
*http://java.sun.com/products/jsp/*

[11] Gnu JSP Servlet Engine
*http://www.klomp.org/gnujsp/*