Das 1400 Byte WAP-Problem

Studienarbeit am Fachbereich Informatik
der Universität Hamburg
– Arbeitsgruppe Verteilte Systeme –
von
Christian Philip Kunze

betreut durch
Prof. Dr. Winfried Lamersdorf

Zusammenfassung

Mobile Kommunikation hat sich in den vergangenen Jahren stetig weiterentwickelt. Mit dem Wireless Application Protocol (WAP) ist eine Architektur entstanden, die verschiedene Inhalte auf tragbaren Geräten verfügbar machen kann. Anwendungen, die nahezu überall verfügbar sein sollen, kommen auf Grund der Verbreitung von Handys kaum noch ohne WAP aus. Durch den Einsatz einer noch sehr jungen Technologie, treten manchmal Probleme auf, die noch gelöst werden müssen. Eines davon ist das *1400 Byte Problem*, welches in dieser Arbeit behandelt werden soll. Neben dem allgmeinen Lösungsansatz wird auf eine spezielle Implementation im Kontext des HYDEPARK-Projekts eingegangen.

Inhaltsverzeichnis

Abbildungsverzeichnis

Tabellenverzeichnis

Kapitel 1

Einleitung

1.1 Motivation

Die Nutzung von mobilen Geräten hat in den vergangenen Jahren stetig zugenommen. Heutzutage sind Handy und der PDA ständige Begleiter vieler Menschen geworden. Daher lag es nahe, diese Geräte mit mehr Funktionalität auszustatten. Jedoch stößt man schnell an die Grenzen der technischen Möglichkeiten, die durch eine im Vergleich zu stationären PCs sehr geringe Speicherkapazität und Rechenleistung begründet ist. Auch die Datenübertragungsraten sind in den heute zur Verfügung stehenden GSM[1]-Netzen mit 9.600 bps noch sehr gering. Technische Neuerungen wie GPRS[2] oder der neue Mobilfunkstandard UMTS[3] sollen diese Übertragungsrate verbessern, setzen sich jedoch nur langsam durch. Weitere Probleme gibt es durch die verhältnismäßig kleinen Displays und die eingeschränkten Eingabemöglichkeiten. Eine der Neuerungen, die heutige Handys bieten, ist die Möglichkeit, sich mit dem Internet zu verbinden und dort Daten abzurufen. Hierzu wurde vom WAP-Forum – einem neu geschaffenen Normierungsgremium – das Wireless Application Protocol (WAP) entwickelt, welches nicht ein einzelnes Protokoll darstellt, sondern ein ganzer Protokollstapel ist. Auf Anwendungsebene wurde die Textauszeichnungssprache WML (Wireless Markup Language) entworfen, mit der man, ähnlich wie mit HTML[4] für Browser am heimischen PC, Seiten für den WAP-Browser des mobilen Geätes schreiben kann.

Durch WAP ist es möglich über Intranets oder das Internet auf Daten, die auf dem Computer am Arbeitsplatz liegen, zuzugreifen. Führt man diesen

[1] Global System for Mobile Communication
[2] General Packet Radio Service
[3] Universal Mobile Telecommunications System
[4] HyperText Markup Language

Abbildung 1.1: Technische Grenzen von mobilen Geräten (nach [2])

Ansatz weiter, so ist der nächste Schritt der, daß man auch von unterwegs auf Anwendungen zugreifen möchte. Um dies zu ermöglichen wird in dem HYDEPARK-Projekt der *Arbeitsgruppe Verteilte System* eine Softwarearchitektur entwickelt, die es ermöglicht, plattformunabhängig auf JAVA-Anwendungen zuzugreifen. Hierbei wird die Benutzungsschnittstelle zur Laufzeit beobachtet und nach Anforderung in eine abstrakte Beschreibungssprache übersetzt. Da diese Sprache auf XML[5] basiert, läßt sich nun mit Hilfe verschiedener XSL-Stylesheeds, je nach Endgerät, eine HTML oder WML Seite erstellen, um sie an den Client zu übertragen. Änderungen in der Benutzungsschnittstelle werden reziprok verarbeitet und so dem Programm wieder zugänglich gemacht.

1.2 WAP im Überblick

Mit dem Wireless Application Protocol will das WAP-Forum zwei sich zur Zeit stark entwickelnde Netzwerktechnologien zusammenführen. Dies sind das Internet und die Mobilkommunikation[6]. Bei diesem Versuch sahen sich die Mitglieder des Forums den schon in Kapitel 1.1 dargestellten Limitationen gegenübergestellt und erkannten, daß die im Internet verwendeten Protokolle nicht ohne weiteres übertragen werden können. Deshalb entschloß man sich, eine eigene Protokollstruktur zu entwickeln, die dort, wo es möglich ist, bestehende Standards verwendet. Andere Teile des Protokollstapels finden zwar Entsprechnungen in bestehenden Technologien, sind aber nicht vollständig kompatibel zu ihnen. Dabei haben die einzelnen Schichten folgende Aufgaben (vergleiche Tabelle 1.1): Die sogenannten *Bearers* bezeichen die zu

[5] Extensible Markup Language
[6] vergl. [15] Seite 9ff

Grunde liegende Netztopologie, welche für die Wegeermittlung, die Geräteadressierung sowie optional für die Fehlererkennung, die Segmentierung und das Wiederzusammenfügen Verantwortung trägt. Die nächste Schicht ist der Datagramm Dienst des Protokollstapels und wird als *Wireless Datagram Protocol* bezeichnet. Die Hauptaufgabe dieses Dienstes ist es, eine einheitliche Schnittstelle auf die verschiedenen Netztopologien aufzusetzten[7]. Dabei beinhaltet WDP die Port Adressierung sowie - auch hier optional - Fehlererkennung und Segmentierung mit Wiederzusammenfügen. Nach WDP kommt *WTLS*, die Wireless Transport Layer Security Schicht. Sie hat die Aufgabe, eine Kommunikation zwischen zwei Systemen durch eine entsprechende Verschlüsselung abzusichern. Das *Wireless Transaction Protocol* ist für den Umgang mit Transaktionen ausgelegt. Diese Schicht bietet drei verschiedene Arten von Transaktionen an, die sich hauptsächlich in der Anzahl der versendeten Bestätigungen unterscheiden. Dabei müßen verlorene und duplizierte Sendungen erkannt und entsprechende Reaktionen eingeleitet werden. Nach dieser Schicht folgt das *Wireless Session Protocol*, das die Aufgaben hat, eine zuverlässige Sitzung zwischen Client und Server einzurichten, Fähigkeiten zwischen ihnen abzusprechen, Inhalte zu versenden und wenn nötig, die Sitzung zu unterbrechen und ggf. wieder aufzunehmen[8]. Die *Wireless Application Environment* bildet den Abschluß des Protokollstapels und ist der Teil, mit dem der Benutzer in Berührung kommt. Hier ist unter anderem die zur Gestaltung der Seiten zu verwendende Sprache WML definiert wie auch der dazugehörige Browser.

Das vom Internet bekannte Modell, bei dem sich ein Client direkt von einem Server die von ihm benötigten Daten beschafft, ist in der WAP Architektur erweitert worden. Zwischen Client und Server befindet sich noch das WAP Gateway, das die Aufgabe hat, die Internetprotokolle in die WAP-Protokolle bzw. vice versa zu übersetzen[9]. Das WAP Modell ist in Abbildung 1.2 noch einmal grafisch verdeutlicht. Das Gateway hat seinen Einsatzort vor allem bei Serviceprovidern und den Betreibern der Mobilfunknetze. Aber auch Unternehmen können ihre Intranets mit Hilfe des WAP Gateways mobil verfügbar machen.

1.3 JBSA im Überblick

Die beschriebenen Limitationen mobiler Geräte machen es schwierig,komplexe Anwendungen für diese zu schreiben. Trotzdem möchte man häufig

[7] vergl. [15] Seite 17
[8] vergl.[8] Kapitel 2.5
[9] vergl. [15] Seite 12f

WTP Benutzer z. B. WSP	
WTP	Transaktions Steuerung Senden von Bestätigungen Erkennen von verlorenen und duplizierten Sendungen
WTLS	Verschlüsselung (optional) Überprüfung der Authentizität (optional)
Datagramm Dienst z.B. WDP, UDP	Port Adressierung Segmentierung und Wiederzusammenfügen (optional) Fehlererkennung (optional)
Bearer z.B. IP, GSM SMS/USSD	Wegeermittlung Geräteadressierung Segmentierung und Wiederzusammenfügen (optional) Fehlererkennung (optional)

Tabelle 1.1: Übersicht der Aufgaben einzelner WAP-Schichten

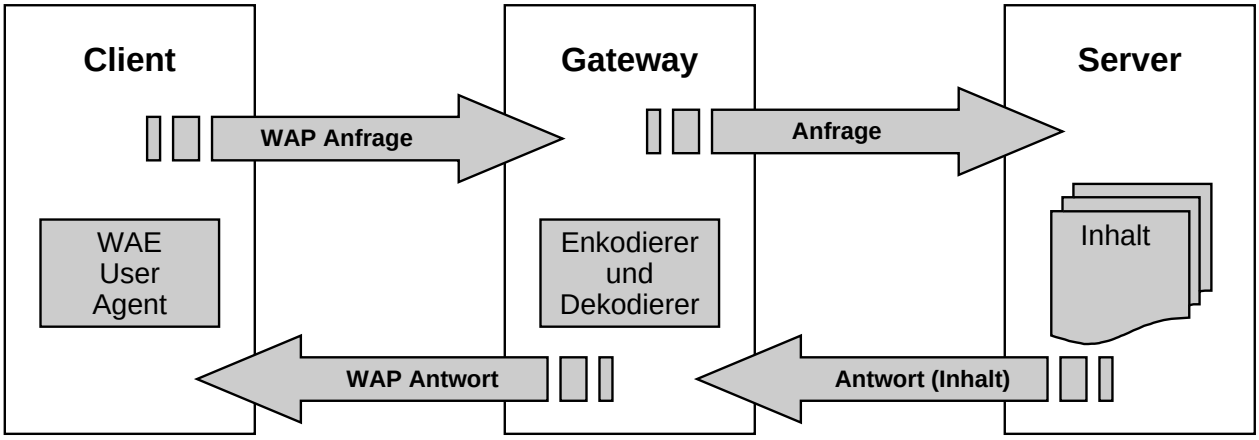

Abbildung 1.2: Das WAP Modell (nach [15])

nicht auf Softwarekomponenten aus dem Büro verzichten. Dieses Dilemma beschäftigt schon seit einiger Zeit viele Entwickler, die auch verschiedene Lösungsansätze gefunden haben. So vertreibt die Firma Citrix mit ICA[10] eine Client/Server-Anwendung, die es erlaubt, die Oberfläche des Hostrechners als graphisches Objekt an die Clients zu übertragen. In dieser Richtung bewegt sich auch das von AT&T entwickelte und frei erhältliche VNC[11]. Beide Verfahren brauchen jedoch client-seitig spezielle und nur für dieses Protokoll entworfene Darstellungsprogramme.

Im HYDEPARK Projekt ist eine Architektur entstanden, die ein größtmögliches Maß an Plattformunabhängigkeit, Geräte– und Applikationstranzparenz bietet. Dadurch kann mobilen Geräten Software zugänglich gemacht werden. Die Plattformunabhängigkeit soll es möglich Machen, mit jedem beliebigen Gerät, das über einen Anschluß an das Internet verfügt, auf Anwendungen zuzugreifen. Es wird durch die Gerätetransparenz vermieden, daß Veränderungen an den mobilen Clients notwendig sind. Auch bei der Programmierung der Software, die durch die HYDEPARK Architektur zugänglich gemacht werden soll, müssen dank der Applikationstranzparenz keine Modifikationen vorgenommen werden. Um dieses leisten zu können, ist die '*JAVA Border Service Architecture*' (JBSA) entwickelt worden[12].

1.3.1 Trennung von Anwendungslogik und Darstellung

Beobachtet man den Aufbau von Programmen, so kann man diese in verschiedene Schichten aufteilen. Es lassen sich zwei grundlegende Einheiten identifizieren. Zum einen die Anwendungslogik, in der Daten verarbeitet werden und deren Repräsentationen angelegt und verwaltet werden. Zum anderen kann man die Darstellung der Daten und die Interaktion mit dem Benutzer an der Benutzungsoberfläche zusammenfassen. Diese beiden Schichten interagieren, um die Leistung der Software zu vollbringen. In der JBSA wird zwischen diese beiden Schichten eine dritte hinzugefügt, die die Elemente der grafischen Oberfläche losgelöst von ihrer tatsächlichen Darstellung aufnimmt. Durch diese Abstraktionsebene ist es möglich, die Benutzungsschnittstelle für verschiedene Geräte aufzuarbeiten. Hierfür bietet sich eine Repräsentation in XML an, da durch verschiedene Stylesheets eine Umformung in andere Sprachen wie HTML oder WML leicht möglich ist. Dabei kommunizieren die Clients über den *ECA*[13] mit dem JBSA Gateway. Die zentrale Aufgabe dieser Einheit ist das Bereitstellen der verschiedenen Kommunikati-

[10] Independent Computing Architecture, vergl. [4]
[11] Virtual Network Computing, vergl. [1]
[12] vergl. [10]
[13] External Communications Adapter

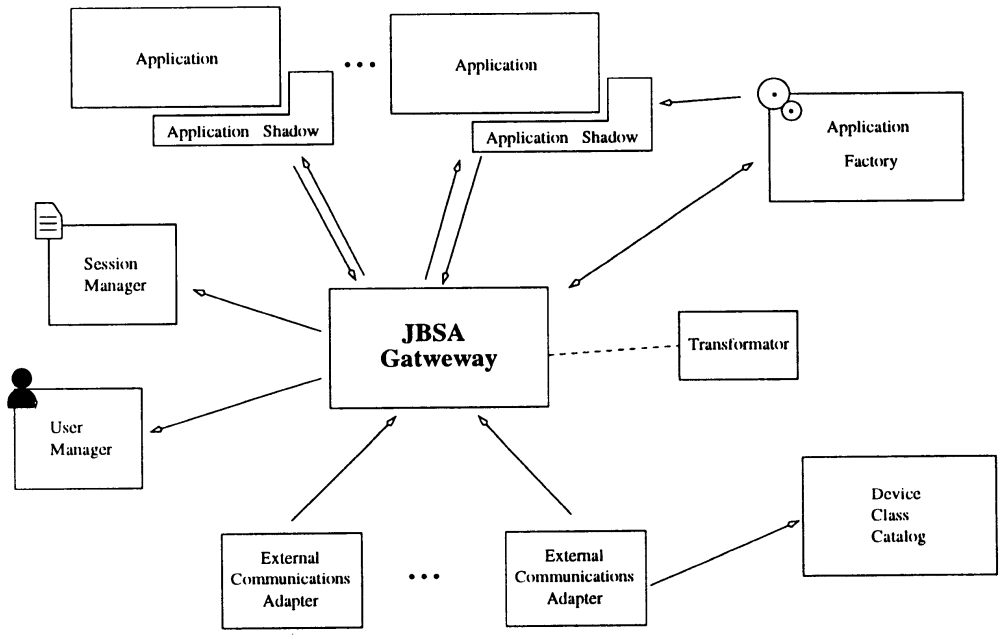

Abbildung 1.3: *JBSA* und ihre Komponenten (aus [10])

onsprotokolle, die die unterschiedlichen Geräte fordern. Das *Gateway* ist die zentrale Koordinierungseinheit der Architektur. Es leitet die von den verschiedenen ECAs übergebenen Ereignisse (zum Beispiel das Drücken eines Buttons oder das Selektieren eines Listeneintrags) zu den korrespondierenden Anwendungen. Eine weitere Aufgabe ist das Ansprechen der verschiedenen XSLT Prozessoren, damit die zur Laufzeit gewonnenen Informationen über das Benutzungsinterface im entsprechenden Format vorliegen. Eine nicht weniger wichtige Aufgabe ist das *Sitzungsmanagement*, das Anwendungen und Benutzer mit einander verbindet. Für jeden Benutzer wird von der *Application Factory* die verlangte Anwendung gestartet und im System registriert. Das Gateway arbeitet nicht direkt mit der Applikation zusammen, sondern mit einem zugehörigen *Application Shadow*. Diese Komponente 'klinkt' sich dabei in die Ereigniswarteschlange der Benutzungsschittstelle ein und zeigt deren Status kontinuierlich an. Neben dem Sitzungsmanagement gibt es noch die *Benutzerverwaltung*, die eine Sicherheitskomponente darstellt und das Accounting und die Überprüfung der Benutzerauthentizität vornimmt. Damit das Gateway überhaupt weiß, welche Stylesheeds verwendet werden müssen und welches die Charakteristika des mobilen Gerätes sind, gibt es den *Device Class Catalog* (DCC). In ihm sind alle notwendigen Daten enthalten und stehen so dem JBSA System zur Verfügung.

1.4 Zielsetzung

Die Aufgabe dieser Studienarbeit ist es, neben der einführenden Darlegung der zugrunde liegenden Problematik und Architekturen, einen Mechanismus in das HYDEPARK-Framework zu integrieren, so daß vom System dynamisch erstellte WML-Seiten fragmentiert werden können. Dies soll ebenfalls zur Laufzeit geschehen und beinhaltet neben dem Segmentierungsalgorithmus auch das Speichern und gezielte Abrufen der einzelnen Teilseiten. Die Segmentgröße soll dabei variabel gehalten werden und so individuell anpassbar sein.

1.5 Vorgehen

Die Struktur dieser Studienarbeit ist folgendermaßen organisiert: Kapitel 2 beschreibt die dieser Arbeit zugrunde liegenden Technologien. Angefangen mit dem Wireless Application Protocol, welches Schicht für Schicht erklärt wird. Danach wird auf die JAVA Border Service Architecture (JBSA), die in dem Projekt HYDEPARK entwickelt worden ist, eingegangen. In Kapitel 3 wird das Segmentieren auf Anwendungsebene und die Verwendung von URL-Filtern beschrieben. Kapitel 4 beschreibt die konkrete Realisierung, die für JBSA entstanden ist. Besonderes Augenmerk liegt dabei auf der `Segmentierer` Klasse. Das Kapitel 5 faßt die Ergebnisse der vorliegenden Untersuchung noch einmal zusammen und gibt einen Ausblick.

Kapitel 2

Basistechnologien

2.1 Wireless Application Protocol

Um der zunehmenden Heterogenität aufkommender Protokolle entgegenzuwirken, gründeten Ericsson, Motorola, Nokia und Phone.com (damals noch Unwired Planet) im Juni 1997 das WAP-Forum. Die Zusammensetzung des Forums sollte möglichst breit gestreut sein, damit weder die Soft- noch die Hardwarefirmen überproportional vertreten sind. Mitte 1999 zählt das WAP-Forum schon über 100 Mitglieder[1] und heute sind es bereits mehrere hundert geworden. Unter ihnen sind die weltgrößten Service Provider, Infrastrukturhersteller und Softwarefirmen, unter anderem auch Microsoft und Oracle.

Die Abbildung 2.1 zeigt den vom WAP-Forum entwickelten Protokollstapel des Wireless Application Protocols. Die Einteilung der einzelnen Schichten und deren Funktionalität orientieren sich an dem ISO OSI Referenzmodell [ISO7498]. Neben diesem Grundmodell versucht das WAP-Forum, möglichst viele schon vorhandene Industriestandards in ihre Architektur zu integerieren. Überall dort, wo es noch keinen entsprechenden Standard gibt oder dieser für die Funkübertragung verändert werden muß, versucht das WAP-Forum mit anderen Standardisierungsgruppen zusammenzuarbeiten. Um den Ansatz der *möglichst breiten Akzeptanz und Praktikabilität* konsequent fortzuführen, wurde versucht, neben den Normierungsgremien auch Expertenmeinungen aus Praxis und Industrie einfließen zu lassen.

2.1.1 Bearer Service

Auf der Netzwerkebene, die die Schichten 1–3 des Referenzmodells umfassen, ist ein zentrales Prinzip die Netzunabhängigkeit. Hierzu wurde WAP

[1] vergl. [16] Seite 2f

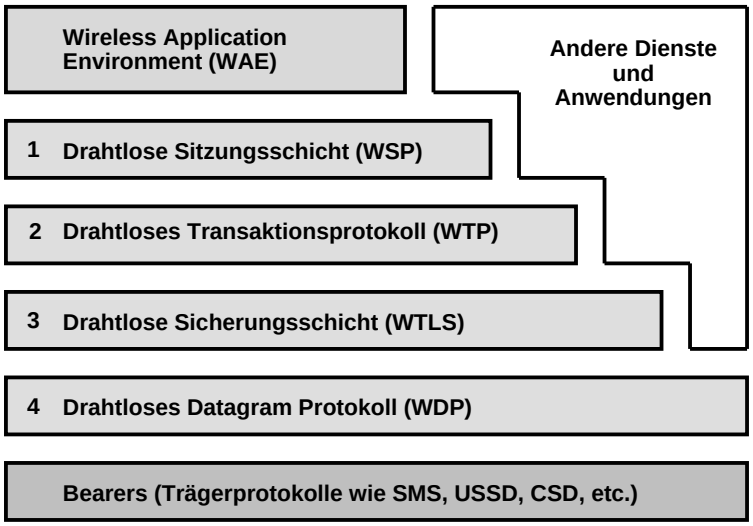

Abbildung 2.1: WAP Protokollstapel (nach[20])

so konzipiert, daß es mit allen drahtlosen Netzen zusammen arbeiten kann. Dies ermöglicht es, einer breiten Masse von Anwendern und Entwicklern über den Protokollstapel auf Anwendungen zuzugreifen bzw. diese zu entwickeln. Damit werden Entwicklungen für die WAP-Architektur interoperabel gehalten, das heißt: Sie sind nicht abhängig von den darunterliegenden Netzdesignentscheidungen und somit problemlos übertragbar.

Aufgrund nur sehr minimaler Anforderungen an das Interface zum Netzwerk kann WAP auch auf Netzen mit extrem geringer Bandbreite, wie dem Short Message Service (SMS) oder dem GSM Unstructured Supplementary Service Data (USSD) Kanal, betrieben werden[2]. Überall dort, wo bereits ein UDP / IP -Stack betrieben wird, kann dieser ebenfalls als Netzschicht verwendet werden.

2.1.2 Wireless Datagram Protocol

Mit dem Wireless Datagram Protocol (WDP) und dem Wireless Transaction Protocol (WTP) hat das WAP-Forum seine Implementierung der Transportschicht (Schicht 4 im ISO OSI Modell) umgesetzt[3]. Mit WDP steht den darüberliegenden Protokollen eine einheitliche Schnittstelle zur Verfügung, um ihre Dienste erbringen zu können. Damit wird das verfügbare Übertragungsverfahren ab dieser Schicht transparent gehalten. Zu den Aufgaben

[2] vergl. [16] Seite 5
[3] vergl. [18] Seite 5

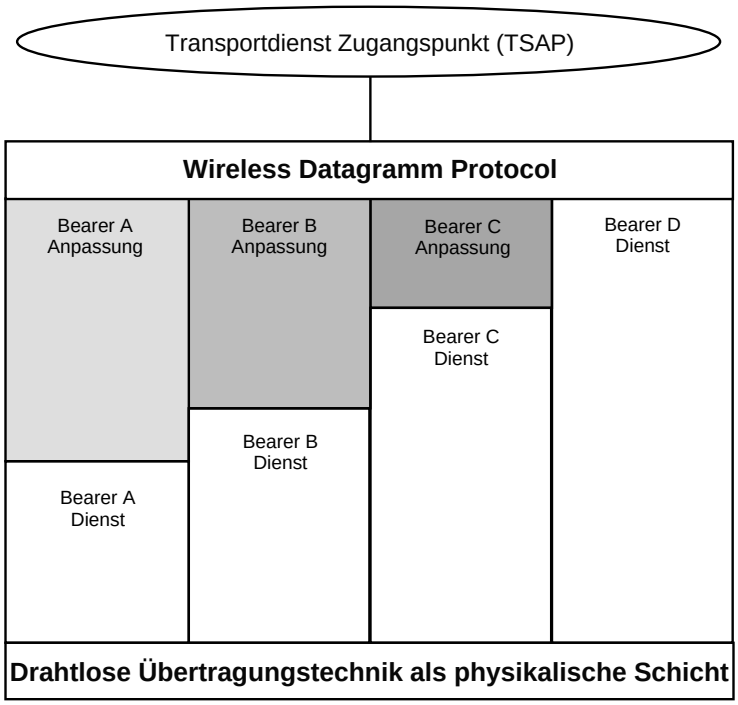

Abbildung 2.2: WDP Architektur (nach [18])

dieses Protokolls gehörten das Adressieren via Port-Nummern, optionales Segmentieren und Wiederzusammenfügen und ebenfalls optional die Fehlererkennung[4].

Wie die Abbildung 2.2 der WDP Architektur zeigt, sind neben der Definition der Transportschittstelle auch Anpassungen an die verschiedenen Netze notwendig. Diese Adaptionen sind ebenfalls in der Spezifikation enthalten, so daß sich die Transportschicht nach außen netztransparent darstellt.

Eine weitere sehr wichtige Aufgabe des WDP-Services ist die Segmentierung und das Wiederzusammenfügen von Protokolldateneinheiten (PDU) überall dort, wo es das darunterliegende Netz fordert. Eine Aufteilung von PDUs ist immer dann notwendig, wenn die zu übertragenden Datagramme größer sind, als die Paketgröße des in Anspruch genommenen Dienstes. Hierbei werden die Daten in eine Sequenz von Protokolldateneinheiten zerlegt, die dann jeweils der Größe der Dienstprotokolldateneinheit entsprechen. So ist die maximale Paketgröße bei der Übertragung über GSM SMS auf 140 Bytes und bei GSM USSD auf 160 Bytes begrenzt[5]. Diese beiden Zahlen machen

[4] vergl. [18] Seite 17
[5] vergl. [18] Seite 65

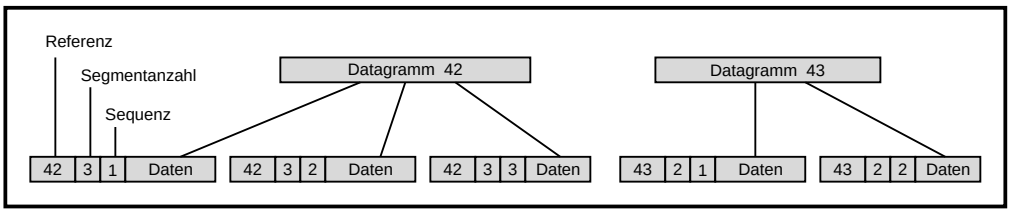

Abbildung 2.3: Segmentierung im WDP (nach [18])

die Notwendigkeit der Segmentierung deutlich; denn durch den mitgeführten Overhead stehen noch weniger als die theoretisch verfügbaren Bytes zur Verfügung. Ein typisches Beispiel für die Segmentierung ist in der Abbildung 2.3 dargestellt. Wichtig ist, daß das Protokoll hierbei mit duplizierten, verlorenen und unsortierten Dateneinheiten umgehen kann, da der vorausgesetzte Dienst als verbindungslos[6] angenommen wird. Das Wiederzusammenfügen wird dann an Hand der mitgeführten Referenz- und Sequenznummern durchgeführt.

2.1.3 Wireless Transport Layer Security

Bei der Spezifikation des Security Layer hat das WAP-Forum auf das Transport Layer Security Protokoll[7] (TLS) in der Version 1.0 zurückgegriffen und dieses als Wireless Transport Layer Security (WTLS) an die Bedürfnisse der drahtlosen Kommunikation angepaßt[8]. Dieses Sicherheitsprotokoll ist optional und setzt auf der Transportschicht auf. Die Aufgaben sind dabei:

- Datenintegrität
  Hierdurch wird sichergestellt, daß Datagramme, die zwei Endsysteme austauschen, auf dem Weg zwischen ihnen nicht verfälscht oder beschädigt werden[9].

- Geheimhaltung
  Die Geheimhaltung hat die Aufgabe dafür zu sorgen, daß ausgetauschte Dateneinheiten nur für die kommunizierenden Endsysteme zugänglich sind. Dies wird durch eine entsprechende Verschlüsselung der zu übertragenden Protokolleinheiten realisiert.

[6] vergl. 2.1.5
[7] TLS ist auch als Secure Sockets Layer Protokoll (SSL) bekannt.
[8] vergl. [15] Seite 16 und [6] Seite 14
[9] vergl. [15] Seite 16

- Überprüfung der Autentizität
  Mit der Überprüfung der Autentizität wird die Identifizierung von Client und Host bezeichnet, so daß jeder der Kommunikationsteilnehmer sicher sein kann, daß sein Gegenüber auch der ist, für den er sich ausgibt.

2.1.4 Wireless Transaction Protocol

Das Wireless Transaction Protocol setzt auf dem datagrammorientierten Dienst des Protokollstapels auf und stellt ein leichtgewichtiges transaktionsorientiertes Protokoll dar. Es ist speziell auf die Bedürfnisse von mobilen Geräten zugeschnitten. Hierbei wird als Transaktion die Kommunikation zwischen einem Initiator der Anfrage und dem Antwortenden bezeichnet.[10] Das WTP kennt dabei drei Klassen von Transaktionsdiensten[11]:

- Unzuverlässige Einweganfrage (Class 0)
  Diese Klasse soll den Transaktionsservice so erweitern, daß Applikationen gelegentliche Datagramme in einem bereits bestehenden Kontext senden können. Diese Erweiterung kann zum Beispiel von Anwendungen genutzt werden, die einen unzuverlässigen Push-Dienst verwenden möchten. Mit Push ist ein Dienst bezeichnet, der automatisch Inhalte an einen Benutzer sendet[12]. Die Class 0 Anfragen werden vom Antwortenden nicht bestätigt und enden für den Initiator nach dem Absenden und für den Empfänger bei Erhalt der Nachricht. Bei einem eventuellen Verlust des Datagrammes wird dieses nicht erneut übertragen.

- Zuverlässige Einweganfrage (Class 1)
  Bei diesem Dienst wird auf die Anfrage eines Initiators vom Angerufenen eine Quittungsnachricht zurückgesendet. Sollte die Antwort ausbleiben, da sie verlorengegangen ist oder ein beim Absender gesetzter Timer abläuft, so wird die Anfrage erneut abgeschickt. Mit diesem Dienst kann ein zuverlässiger Push-Dienst etabliert werden.

- Zuverlässige Zweiweganfrage (Class 2)
  Die Transaktionen dieser Klasse bilden den meist benutzten Dienst dieser Schicht. Eine WSP Sitzung besteht im allgemeinen aus mehreren Transaktionen diesen Types. Hierbei wird auf die Anfrage des Initiators eine Quittung des Angerufenen erwartet. Diese Quittung wird

[10] vergl. [6] Seite 15
[11] vergl. [21] Seite 12ff
[12] vergl. [12] – For a definition of push

dann ebenfalls durch den Auslöser der Transaktion bestätigt. Sollte eine Quittung ausbleiben, so wird diese vom entsprechenden Kommunikationspartner durch erneutes Absenden seines letzten Datagramms angefordert. Diese Transaktion kann von beiden jederzeit abgebrochen werden.

Um die Anzahl der ausgetauschten Nachrichten zu reduzieren, bietet diese Schicht die Möglichkeiten, Protokoll Dateneinheiten (PDU) zu verknüpfen und Quittungsnachrichten zu verzögern[13].

2.1.5 Wireless Session Protocol

Diese Schicht des Protokollstapels bildet das Äquivalent zu der Kommunikationssteuerungsschicht[14] des ISO OSI Referenzmodells. WSP bietet dabei zwei Interfaces an, damit sein Dienst in Anspruch genommen werden kann. Zum einen kann auf ein verbindungsorientierten Dienst zugegriffen werden, der auf der Transaktionsschicht des Protokolls (WTP) arbeitet. Desweiteren wird ein verbindungsloser Dienst angeboten, der den sicheren oder unsicheren Datagramm-Dienst des WAP-Protokollstapels (WDP) benutzt[15]. Als verbindungslos wird eine Übertragung dann bezeichnet, wenn es keine taktische Phase des Verbindungsaufbaus und -abbaus gibt, sondern Daten ohne diese Vorbereitung gesendet und empfangen werden[16]. WSP dient also zum organisierten Austausch von Inhalten zwischen kooperierenden Client/Server Anwendungen. Im speziellen unterstützt es Applikationen in vierfacher Art[17]:

- Etablieren einer zuverlässigen Sitzung zwischen Client und Server.
- Abstimmen allgemeiner Protokollfunktionaliät mittels Fähigkeitsabsprachen.
- Austausch von Inhalten unter Verwendung einer kompakten Kodierung.
- Unterbrechung und Wiederaufnahme von Sitzungen.

Auf dieser Basis begründen sich die angesprochenen Protokollvarianten einer verbindungsorientierten und einer verbindungslosen Sitzung. Eine solche Sitzung kann über einen sehr langen Zeitraum erhalten bleiben. Selbst,

[13] vergl. [15] Seite 16
[14] Die Kommunikationssteuerungsschicht ist die 5. im Modell des OSI Netzwerkes.
[15] vergl. [19] Seite 16
[16] vergl. [7] Seite 142.
[17] vergl. [19] Seite 12

wenn sie unterbrochen wurde, weil zum Beispiel die Funkstrecke Störungen aufweist, kann die Sitzung wieder aufgenommen werden. Eine weitere Fähigkeit von WSP ist es, Sitzungen zwischen verschiedenen Clients migrieren zu können. Neben den generellen Merkmalen unterstützt die WAP Kommunikationssteuerungsschicht auch folgende Merkmale:

- Die HTTP/1.1 Funktionalität[18].
  - Erweiterte Request-Reply Methoden
  - Zusammengesetzte Objekte
  - '*Content type* '[19] Vereinbarungen
- Der Austausch der Sitzungsheader von Client und Server.
- Das Unterbrechen von in Ausführung befindlichen Transaktionen.
- Ein asynchroner Pushdienst vom Server zum Client.

2.1.6 Fähigkeitsabsprachen

Die Möglichkeit, Fähigkeiten durch das Protokoll zwischen den Kommunikationspartnern abstimmen zu lassen, bietet die Chance sehr flexibel auf verschiedene Konfigurationen reagieren zu können. Es entstehen jedoch neue Probleme, da Parameter vorab nicht definitiv bekannt sind. Als Fähigkeit (*engl. Capability*) definiert das WAP-Forum in [19] die Möglichkeiten und Konfigurationsparameter des WSP-Protokolls, die ein Client oder Server unterstützt. Der Abgleich der Fähigkeiten geschieht dabei während des Sitzungsaufbaus und gibt dem Server, die Möglichkeit zu entscheiden, ob der Client Protokollausprägungen und –konfigurationen unterstützt. In eine Fähigkeitsabsprache sind alle an der Kommunikation beteiligten Partner, also neben Client und Server auch die beteiligten Gateways, involviert. In der Protokollspezifikation ist nur die '*one-way capability negotiation*' definiert. Dabei schlägt der Initiator einen Satz von Fähigkeiten vor und der Antwortende geht dann auf diese ein. Dabei geht der Prozeß des Aushandelns folgendermaßen von statten:

1. Der Initiator des Dienstelements schlägt einen Satz von Fähigkeiten vor.

[18] vergl. [14]
[19] Angabe über die Art des zu übertragenden Inhalts.

2. Der Serviceprovider des Initiators verändert die Werte so, daß sie nicht höhere Einstellungen enthalten, als er verarbeiten kann.

3. Auch der Serviceprovider des Erwidernden führt Modifikationen seinen Vorgaben entsprechend durch.

4. Der Erwidernde führt seine Änderungen an den erhaltenen Werten durch und sendet nun diesen Satz an Fähigkeiten zurück. Die Einstellungen repräsentieren den Status, den er verwenden möchte.

5. Damit sind die ausgesuchten Werte an den Initiator zurückgegeben und werden zu den Vorgaben für den Rest der Sitzung.

Sollte dieser Prozeß fehlschlagen, so bleiben die bisherigen Werte in Benutzung. Bei positiven Ganzzahlen wird als Fähigkeit stets nur das Minimum der Werte von Initiator, Service Providern oder Antwortenden unterstützt.

Die in Tabelle 2.1 aufgezählten Fähigkeiten müssen von Benutzer und Dienststeller vorgehalten werden. Für diese Studienarbeit ist besonders die '*Client SDU Size* ' von Bedeutung. Mit dieser wird vom mobilen Geräten mitgeteilt, wie groß eine einzelne übertragene Dateneinheit an es sein darf. Dieser Wert wird maßgeblich durch die Speicherkapazität des Clients bestimmt. Die WSP Spezifikation definiert für dieses Element, wie auch für alle anderen Fähigkeiten, einen Standardwert, der in diesem Fall bei 1400 Bytes liegt[20]. Dieser Wert ist frei verhandelbar und darf durchaus auch den Standard nach oben hin durchbrechen, solange alle beteiligten Diensterbringer und –benutzer diese Größe unterstützen.

2.1.7 Wireless Application Environment

Mit der Wireless Application Environment hat das WAP-Forum eine für vielfältige Zwecke nutzbare Anwendungsumgebung entwickelt, die auf der Philosophie und Technologie des World Wide Webs basiert. Die WAE soll eine interoperable Umgebung sein, die es Service Providern und Entwicklern gestattet, Anwendungen und Dienste zu gestalten, die auf einer Vielzahl verschiedener Plattformen effizient ablaufen. Dabei wurden vier Ziele angegangen:[21]

- Definition eines Anwendungsarchitekturmodells
  Dieses Modell soll in die bestehende WAP-Architektur passen und

[20] vergl. Tabelle 2.1.6 und [19] Seite 72
[21] vergl. [20] Seite 10f

Fähigkeit	**Klasse**	**Typ**	**Beschreibung**
Aliase	I	Adressen-liste	Hier kann ein Service Provider Alternativadressen angeben, mit denen die gleiche Dienstbenutzerinstanz erreicht werden kann. Dabei sind die Angaben prioritär geordnet.
Client SDU Größe	N	positive Ganzzahl	Angabe über die maximale Größe der Transaktions SDU, die an den Client gesendet werden darf.
Erweiterte Methoden	N	Satz von Methoden-namen	Hier können erweiterte Methoden abgesprochen werden, die neben denen von HTTP/1.1 verwendet werden sollen.
Header Code Pages	N	Satz von Code Page Namen	Hier können erweiterte Header Code Pages abgesprochen werden, die von Client und Server unterstützt werden.
Maximal ausstehende Methodenanfragen	N	positive Ganzzahl	Maximale Anzahl von Methodenaufrufen, die gleichzeitig aktiv sein können.
Maximal ausstehende Push-Anfragen	N	positive Ganzzahl	Maximale Anzahl der ausstehenden Pushbestätigungen.
Protokoll Optionen	N	Satz von Ausprägungen	Hiermit werden optionale Dienstelemente aktiviert. Dabei kann aus folgender Liste gewählt werden: Push, Bestätigtes Push, Sitzungswiederaufnahme und Bestätigungs Header
Server SDU Größe	N	positive Ganzzahl	Angabe über die maximale Größe der Transaktions SDU, die an den Server gesendet werden darf.

N steht für aushandelbar und I für informativ.

Tabelle 2.1: Definierte Fähigkeiten im WSP

Name	Wert
Aliase	Keine
Client SDU Size	1400 Bytes
Erweiterte Methoden	Keine
Header Code Page	Keine
Protokoll Optionen	0x00
Maximal Ausstehende Methodenanfragen	1
Maximal Ausstehende Push-Anfragen	1
Server SDU Size	1400 Bytes

Tabelle 2.2: Standardeinstellungen der WSP Fähigkeiten

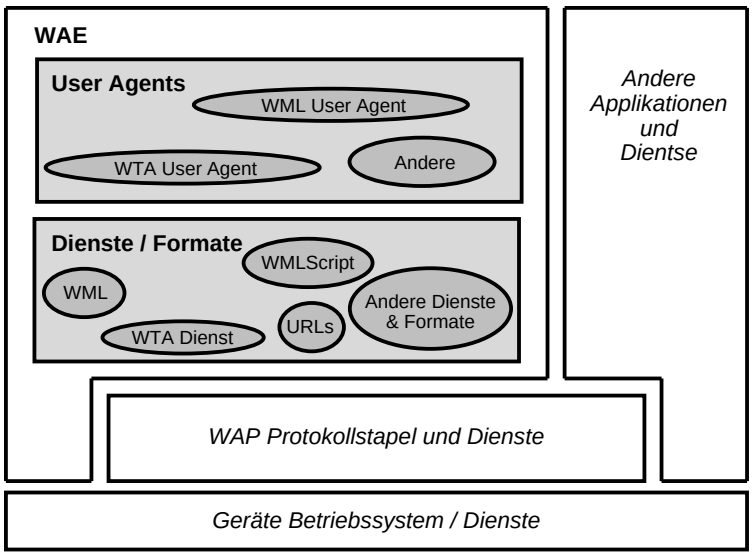

Abbildung 2.4: WAE Client Komponenten (nach [20])

es ermöglichen, interaktive Applikationen zu entwickeln, deren Funktion gut mit den bekannten Limitationen mobiler Geräte umgehen kann. Dabei sollen Sicherheits- und Zugriffskontrollen eine unkritische Ausführung von fremden Inhalten erlauben. Das Modell soll global einsatzfähig sein und Internationalisierungstechniken unterstützen.

- Definition eines Mehrzweckprogrammiermodells
  Das Programmiermodell soll vielfältig sein und interaktive Anwendungsentwicklung mit heutigen und zukünftigen drahtlosen Geräten möglich machen. Dabei sollen sich die Elemente zur Entwicklung an den im Internet bekannten Modellen des Browsings und Scriptings orientieren. Aber auch der Zugriff auf typische Inhalte von tragbaren Geräten wie Telefonbuch oder Mitteilungsdienst soll bereitgestellt werden.

- Bereitstellen von Netzwerkoperationen
  Die Operationen sollen dabei die bestehenden Netzwerkdienste erweitern und verbessern.

- Ermöglichen von Interoperabilität zwischen verschiedenen Implementierungen der WAE
  Hierdurch soll das Ziel von wiederverwendbaren Anwendungen angestrebt werden. Alle für eine Plattform entwickelten Applikationen sollen auf allen anderen ebenfalls problemlos laufen.

Aus diesen Zielen sind zwei Hauptelemente der WAE hervorgegangen. Zum einen die '*User Agents* ' und zum anderen die '*Services und Formate* '. Dies wird in der Abbildung 2.4 graphisch verdeutlicht.

WAE User Agents

Hierbei handelt es sich um client-seitige Softwarekomponenten, die spezielle Funktionen für den Endbenutzer bereitstellen. Ein Browser, der WML Seiten anzeigen kann, ist ein User Agent. Diese Komponenten sind in die WAP-Architektur integriert und interpretieren einen über eine URL[22] referenzierten Netzwerkinhalt. Es gibt dabei zwei Standardinhalte, die Wireless Markup Language und die zugehörige Scriptsprache WMLScript. Über die Erzeugung dieser Inhalte macht die WAE keine Aussagen. Es wird jedoch davon ausgegangen, daß alle Dienste, die auf HTTP Servern laufen (zum Beispiel CGI[23] oder JAVA-Servlets), auch WAE Inhalte generieren können.

[22] Uniform Resource Locator
[23] Common Gateway Interface

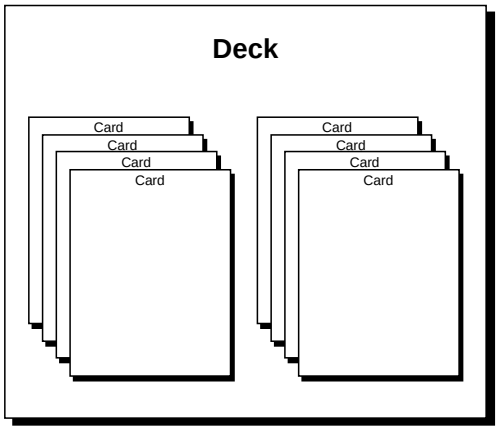

Abbildung 2.5: WML Deck und Cards (nach [11])

Ebenfalls enthalten ist eine Sammlung von telefonspezifischen Erweiterungen. Diese Mechanismen zur Anruf- und Telefonkontrolle sind unter dem Begriff Wireless Telephony Applications (WTA) zusammengefaßt und sollen den Entwicklern erweiterte Dienste des mobilen Netzwerkes zur Verfügung stellen.

WAE Services und Formate

Die WAE kennt verschiedene Formate, die es den User Agents möglich machen, durch Webinhalte zu navigieren. Dazu gehören WML und WMLScript, Standardbildformate und Containerformate zum leichten Datenaustausch.

Die auf XML basierende Textauszeichnungssprache WML wird, wie HTML im World Wide Web, dazu verwendet, Netzinhalte zu formatieren. Dabei sind den besonderen Bedürfnissen in Mobilnetzen Rechnung getragen worden. Informationen werden in Sammlungen von '*Cards* ' und '*Decks* ' organisiert. Cards enthalten dabei die Interaktionen mit dem Benutzer wie zum Beispiel ein Auswahlmenü, einen Text oder Eingabefelder. Ein Anwender navigiert durch eine Reihe von WML Cards, sieht sich deren Inhalte an, macht selbst Eingaben oder führt eine Auswahl durch und geht zu einer weiteren Card über[24].

Verschiedene Cards werden dann in den Decks, wie in Abbildung 2.5 zu sehen, miteinander gruppiert. Ein Deck ist in WML die kleinste Einheit, die ein Server an den User Agent übertragen kann. Decks können ein Template-Element enthalten, das Charakteristika beschreibt, die in allen Cards inner-

[24] vergl. [11] Seite 17

```
<?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="MyCard">
      <p>
         Hello World !
      </p>
   </card>
</wml>
```

Abbildung 2.6: '*Hello World !*' als WML Dokument

halb des Verbundes enthalten sein sollen[25]. Als drittes Element kann ein Deck noch ein Head enthalten, in dem neben dem Autor und ähnlichen Angaben auch eine Zugriffskontrolle enthalten sein kann. Ein einfaches WML-Dokument ist in Abbildung 2.6 als Beispiel gegeben.

WMLScript ist eine einfache prozedurale Scriptsprache. Sie erweitert WML um weitere Möglichkeiten der Präsentation und Benutzerinteraktion. Über die Scriptsprache kann auch auf erweiterte Dienste des Gerätes und seiner angeschlossenen Peripherie zugegriffen werden. WMLScript basiert dabei lose auf der im World Wide Web verbeiteten Sprache JAVAScript, ist aber speziell an die Bedürfnisse der drahtlosen Kommunikation angepaßt[26].

Ein weiteres von der WAE definierte Format ist der Grafikstandard des Wireless Bitmaps (WBMP). Dieses wird dazu verwendet, Schwarzweißgrafiken in den WML User Agents darzustellen (siehe Abbildung 2.7). Daneben gibt es noch Containerformate, die den standardisierten Austausch von Business- oder Kalenderdaten – wie zum Beispiel elektronische Visitenkarten oder Terminabsprachen – ermöglichen sollen.[27].

2.2 Funktionalität der JBSA

Nach dem ein Überblick der verschiedenen Komponenten der JBSA-Architektur in Kapitel 1.3 gegeben wurde, soll in desem Abschnitt der typische Zugriff eines mobilen Gerätes auf eine Anwendung via JBSA gezeigt werden. Dabei wird auf die erkennbaren Schritte und die beteiligten Komponenten

[25] vergl. [11] Seite 17 und 19
[26] vergl. [20] Seite 19
[27] vergl. [20] Seite 20f

Abbildung 2.7: Das Fachbereichslogo als WBMP

sowie deren Arbeitsweise eingegangen. Dabei können nach [10] vier Schritte unterschieden werden.

- Kontakt herstellen
  Die Geräte verbinden sich mit dem Extenal Communication Adapter über einen geeigneten Zugangspunkt. Dieser kann zum Beispiel eine Internet-Adresse oder eine Telefonleitung sein. Um die Klasse des Gerätes zu ermitteln, wird eine Anfrage vom ECA an den Device Class Catalog gestellt. Die Antwort wird zusammen mit dem für dieses Verbindung eingerichteten Zugang zum Gateway in einer Gerätesitzung gespeichert. Hiernach wird eine Anfrage an das JBSA-Gateway geschickt, damit dieses eine Applikationssitzung für die Gerätesitzung einrichtet.

- Initialisierung einer Applikationssitzung
  Zunächst wird vom JBSA-Gateway ein Login-Dialog erzeugt und in das für das Gerät passende Format überführt. Danach wird dieser an den ECA zurückgegeben. Der ECA leitet das Dokument an das Gerät weiter, damit die zur Authentifizierung nötigen Eingaben gemacht werden können. Die Eingaben werden dann via ECA an das Gateway zur weiteren Verarbeitung übergeben. Die Autentifizierungsdaten werden an den User Manager gesand, damit deren Gültigkeit geprüft werden kann. Ist ein Benutzer identifiziert, werden die für die Startphase benötigten Daten (z.B. die Standardapplikation, Parameter, etc.) an das JBSA-Gateway gesendet. Das Gateway fragt dann beim Session Manager an, ob es für den Benutzer unterbrochene Sitzungen gibt. Dabei sieht der Session Manager in seiner Datenbank nach und berichtet über alle An-

wendungen, die mit der Benutzer-Id verbunden sind. Liegen unterbrochene Anwendungen vor, so wird ein weiterer Dialog erzeugt, der es erlaubt diese wieder aufzunehmen, zu beenden oder eine neue Anwendung zu starten. Das entstandene Dokument wird über den ECA an das Gerät übermittelt. Die Auswahl des Benutzers geht zurück an das Gateway. Liegt keine unterbrochene Anwendung vor oder wünscht der Benutzer eine neue zu starten, so wird die Applikationssitzung zusammen mit der vom ECA erzeugten Gerätesitzung im Session Manager registriert.

- Starten der Anwendung
  Da nun eine Sitzung zwischen dem Gerät und der JBSA-Architektur existiert kann eine Anwendung gestartet werden. Um eine Anwendung zu starten, wird die zur Anwendung gehörende eindeutige Kennung der Standardanwendung, der Anwewendung aus den Device Cass Catalog oder einer ausgewählten Applikation an die Application Factory übertragen. Diese sucht in ihrer Datenbank den Pfad und die Parameter der Anwendung. Diese Angaben werden mit einer Startkennung an einen neuen Application Shadow, der von der Factory erzeugt wurde, weitergeleitet. Dieser Application Shadow startet dann die Anwendung in seiner Laufzeitumgebung und verbindet sich mit der zur Anwendung gehöhrenden Ereigniswarteschlange. Diese Verbindung wird für den weiteren Zugriff auf die Anwendung verwendet und mit der Startkennung an das Gateway zurückgegeben. Das JAVA Border Architecture Gateway führt die Applikationssitzung sowie die Startkennung und die Verbindung zur Kommunikation zusammen.

- Benutzen der Anwendung
  Da die Anwendung nun läuft und der Application Shadow bereit ist Momentaufnahmen der Benutzungsschnittstelle zu generieren, kann die eigentliche Interaktion zwischen dem mobilen Gerät und der Anwendung beginnen. Um die erste Momentaufnahme zu bekommen, wird eine entsprechende Anfrage an den zur Anwendungssitzung gehöhrenden Appliction Shadow gesand. Dieser gibt in JSML den Status der Benutzungsschittstelle zurück. Das JSML Dokument wird zusammen mit dem vom Device Class Catalog identifizierten Stylesheet an den XSLT-Prozessor gegeben, der daraus das für das Gerät verständliche Dokument erzeugt. Dieses Dokument wird vom Gateway an den zum mobilen Gerät gehöhrenden ECA übersand. Dem Benutzer wird die Momentaufnahme mittels eines Browsers präsentiert. Mit dessen Hilfe kann nun die Interaktion mit der Anwendung begonnen werden. Die

Eingaben werden dann wieder an den ECA übermittelt, der das eingehende Dokument mit einer Gerätesitzung verbindet und die durch den Benutzer ausgelösten Ereignisse in eine in JBSA verarbeitbare Form transformiert. Beides geht an das Gateway, welches über die korrespondierende Anwendungssitzung die Ereignissse an den richtigen Application Shadow weiterleiten kann. Der Shadow wiederum überführt die JBSA Ereignisse in gültige JAVA SWING Ereignisse und gibt diese an die Anwendung weiter. Die Anwendung kann nicht unterscheiden, ob die Ereignisse vom Shadow erzeugt wurden oder ob sie durch eine lokale Interaktion ausgelöst wurde. Deshalb agiert die Applikation genau so als ob das Programm direkt am PC bedient wird und ändert seine Benutzungsschittstelle entsprechend. Nach einer unmerklichen Verzögerung wird die nächste Momentaufnahme vom Application Shadow angefordert und an das mobile Gerät übermittelt. Dies wird solange weitergeführt, bis der Benutzer die Anwendung beendet oder unterbricht.

Kapitel 3

Lösungsansatz

3.1 Notwendigkeit von Segmentierung auf Anwendungsebene

Im Kapitel 2.1.2 ist gezeigt worden, daß es notwendig sein kann, Dateneinheiten zu segmentieren, bevor man sie überträgt. Dabei wurde gesagt, daß eine Aufteilung von PDUs immer dann notwendig ist, wenn die zu übertragenden Datagramme größer sind als die Paketgröße des in Anspruch genommenen Dienstes. Im Abschnitt 2.1.6 ist verdeutlicht worden, daß auch das Wireless Session Protocol Limitationen der maximalen Größe der zu übertragenden Dateneinheiten unterliegt. Es stellt sich die Frage, warum in dieser Schicht nicht wie in den darunterliegenden Schichten Segmentierung und Wiederzusammenfügen in der Spezifikation enthalten sind? Dies liegt in der andersartigen Ursache der Limitation. In den Schichten unterhalb der Kommunikationssteuerungsschicht werden die Übertragungseinschränkungen von den darunterliegenden Diensten begründet. Bei WSP hingegen wird die Einschränkung der Kapazität durch das darüberliegende Wireless Application Environment verursacht. Da dieses keine größeren Dateneinheiten verarbeiten kann, bringt eine Segmentierung in diesem Fall nichts, da die Datenblöcke beim Empfänger wieder zusammengefügt werden müßten und dieser dies aus einem in ihm begründeten Umstand nicht kann.

Werden zum Beispiel Dokumente, die in WML formatiert sind, übertragen, so muß schon beim Erstellen darauf geachtet werden, daß die Maximalgröße nicht überschritten wird. Aber genau das ist ein Problem, da dieser Wert nicht a priori festliegt, sondern von Client zu Client unterschiedlich sein kann. Noch problematischer sind dynamisch erstellte WML-Seiten, da bei diesen weder deren Umfang noch die Maximalkapazität im voraus bekannt ist. Zur Lösung dieses Problems bietet es sich an, eine Anwendung

zwischen Client und Server zu integrieren, welche die angeforderten Daten in die maximal zulässige Größe segmentiert.

Die maximale Übertragungsgröße ist dabei nicht auf einzelne Arten von Daten beschränkt, sondern gilt für jegliche Übertragung. In dieser Arbeit wird nur auf einen Teil dieses Segmentierungsproblems eingegangen, das im HYDEPARK-Kontext auftritt. Dies ist das Segmentieren von WML-Dokumenten, die dynamisch erzeugt werden.

3.1.1 Segmentierung von WML-Dokumenten

Die Segmentierung von WML-Dokumenten gliedert sich durch die Einteilung in Decks und Cards[1] in zwei Fälle auf. Man könnte die beiden nötigen Vorgehensweisen als Deck- und Cardsegmentierung bezeichnen.

- Decksegmentierung
  Die Decksegmentierung ist dann anzuwenden, wenn der zu fragmentierende Deck mehrere Cards enthält, die zusammen mit der WML- und Deckdeklaration die maximale Byteanzahl überschreiten. Das eigentliche Aufteilen des Decks geschieht dann durch sukzessives herausnehmen einzelner Cards. Diese werden dann in einem eigenen Deck aufgenommen. Dies geschieht solange, bis entweder der Deck mit den verbleibenden Cards kleiner gleich der gesetzten Grenze ist oder sich alle Cards jeweils in einem eigenen Deck befinden. Die Decks, die nun immer noch größer als erlaubt sind, werden noch der Cardsegmentierung unterzogen.

- Cardsegmentierung
  Bei der Cardsegmentierung enthält der zu fragmentierende Deck lediglich eine Card, deren Inhalt zusammen mit den Deklarationen das Limit übersteigt. Bei dieser Art der Segmentierung werden Decks erzeugt, die jeweils einen Teil des Inhaltes der ursprünglichen Card enthalten. Dabei ist zu beachten, daß gültige XML Dokumente erzeugt werden. Dies bedeutet, daß alle Tags, die einmal geöffnet worden sind, auch wieder geschlossen werden müssen. Dies führt dazu, daß man um ein Mitprotokollieren der Tags nicht umhin kommt. Denn alle im Hauptdokument geöffneten Tags müssen zunächst geschlossen werden, um dann im Nachfolgesegment wieder geöffnet zu werden. Nur so bleibt die Formatierung des Inhalts erhalten.

  Durch die Segmentierung kann es dazu kommen, daß es am Ende einer neu geschaffenen Card ein WML-Element gibt, das keinen Inhalt

[1] vergl. 2.1.7

hat. Wenn die Dokumenttyp Definition[2] dies für dieses Element nicht vorsieht, so erzeugt man ein ungültiges Dokument. In einem solchen Fall müßen diese WML-Elemente aus der Card wieder entfernt werden. Elemente, die nicht leer sein dürfen sind : `<select>`, `<optgroup>`, `<table>`, `<tr>`, `<do>` und `<onevent>`[3].

Bei beiden Segmentierungsarten ist es wichtig, daß man darauf achtet, daß die im ursprünglichen Deck eventuell enthaltenen Head- und Templateinformationen auch in die neu zu schaffenden Decks aufgenommen werden. Dies ist nötig, da die im Template enthaltenen Befehle laut WML-Spezifikation für alle Cards innerhalb des Decks gelten und und so implizit auch für die neu geschaffenen Decks gelten müssen. Im Head können Befehle enthalten sein, die zum Beispiel die Zugriffsberechtigung steuern und deshalb ebenfalls Auswirkungen auf die neuen Decksegmente aufweisen.

Nach der Deck- und Cardsegmentierung ist das ursprüngliche WML-Dokument syntaktisch auf mehrere eigenständige Dokumente aufgeteilt. Jedoch müßen einige Tags noch semantisch an die neue Struktur angepasst werden.

3.1.2 Semantische Anpassungen

Auf Grund der Möglichkeit, mehrere Cards in einem Deck zu implementieren, bietet WML eine Erweiterung von Verweisen an, mit der man auf innere Cards zugreifen kann. Dazu wird an die URL zunächst eine Raute (#) und dann der Fragmentidentifizierer angefügt. Beispielhaft soll dies an einem `<go>`-Element gezeigt werden, das auf die interne Card mit dem Identifizierer “Weitere_Card” verweist.

```
<go href="#Weitere_Card"/>
```

Durch die Segmentierung kann es bei solchen Elementen, die einen Verweis auf eine innere Card haben, zu Fehlern kommen, wenn dies nicht durch semantische Korrekturen verhindert wird. Da bei der Segmentbildung zumindest einzelne Cards aus dem Deck entnommen werden und eine eigenständige Einheit in einem neuen Deck bilden, müssen die entsprechenden internen Verweise in externe gewandelt werden. Diese externen Verweise müssen dann auf den Deck zeigen, der den Anfang bzw. die komplette Card enthält, die zuvor intern referenziert wurde. Das `<go>`-Element sieht nach der Anpassung folgendermaßen aus:

[2] engl.Documenttype Definition (DTD)
[3] siehe Anhang C

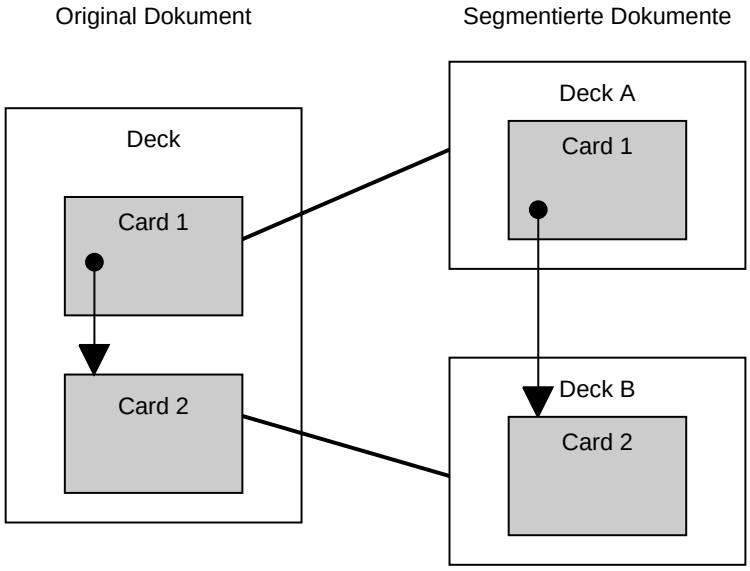

Abbildung 3.1: Wandlung von internen zu extenen Verweisen

```
<go href="http://server/Dokument#Weitere_Card"/>
```

Dies ist in Abbildung 3.1 noch einmal grafisch verdeutlicht. Eine solche Korrektur ist nicht nur bei dem `<go>`-Element nötig, sondern auch bei den Tags `<card>`, `<template>`, `<option>` und `<a>`[4].

3.2 Verwendung eines URL-Filters

Nachdem man WML-Dokumente segmentiert hat, steht man vor dem Problem, die Segmente an den Client zu übertragen. Dieser weiß nicht, daß das Ende der Card nicht das eigentliche Ende ist und weitere Segmente folgen. Das heißt, man muß den Segmenten eine Navigation hinzufügen, um alle Teilseiten ansprechen zu können. Dies kann auf verschiedene Weisen realisiert werden. Zum einen können die neu geschaffenen Seiten in eine statische Struktur eingebunden werden. Dies kann zum Beispiel geschehen, indem die Seiten in einem temporären Verzeichnis unter eindeutigen Namen abgelegt werden. Die Navigation kann dann über diese Namen auf die Seiten zugreifen. Eine elegantere Methode ist die Verwendung von dynamischen Strukturen, da hier die Segmentierung und Navigation kombiniert werden kann. Tech-

[4] siehe Anhang B

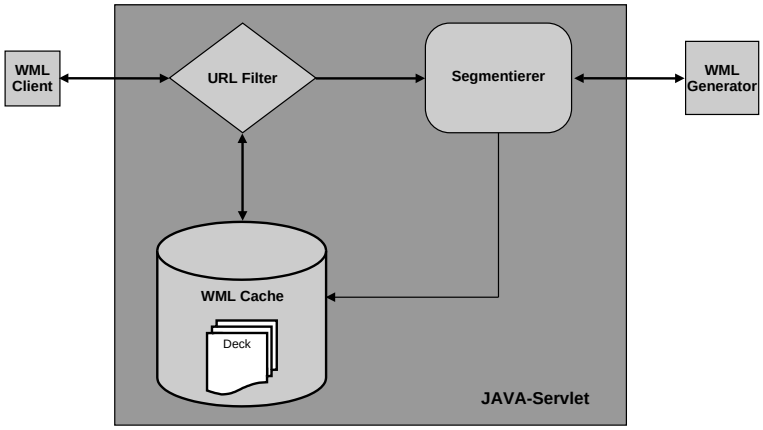

Abbildung 3.2: Schematischer Einsatz eines URL-Filters

nologisch bieten sich hier CGI-Scripts, ASP[5] oder JAVA-Servlets[6] an. Hier wird nicht mit temporären Verzeichnissen und Namen gearbeitet, sondern es kann das Durchblättern der Segmente über in der URL übergebende Parameter gesteuert werden. Eine einfache Navigation könnte einen Parameter mit der Segmentnummer enthalten. Durch Dekrementieren bzw. Inkrementieren kann so eine sequentielle Navigation eingeführt werden. Durch die Angabe einer speziellen Segmentnummer kann aber auch gezielt auf einzelne Teilseiten zugegriffen werden. Die um den Parameter ergänzte URL würde dann folgendermaßen aussehen:

http://server/Dokument**?page=*#num*** $\#num \in [0,1,...]$

Zur Initialisierung der Segmentierung könnte das dynamische Dokument ohne Parameter angesprochen werden. In diesem Fall wird zuerst eine neue Version des Dokumentes angefordert, diese segmentiert und das erste Fragment als Ergebnis an den Client übermittelt. Aufrufe mit Parameter werden dann aus dem Segmentspeicher bedient. Um eine neue Version des eigentlichen Dokuments zu erhalten, könnte wieder ohne Parameter angefragt werden.

Schematisch ist der Aufbau eines solchen URL-Filters in Abbildung 3.2 dargestellt. Hierbei stellt der Client seine Anfrage an das Servlet, der URL Filter entscheidet dann, ob eine neue Segmentierung angestoßen wird oder ob ein Segment aus dem Cache geholt werden muß.

[5] Active Server Page von Microsoft
[6] vergl. [9]

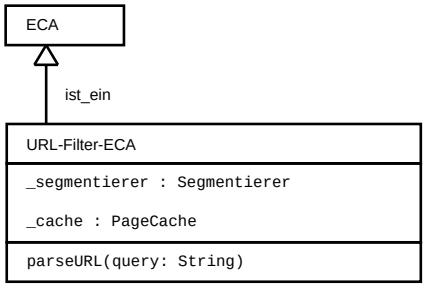

Abbildung 3.3: Klassendiagramm für die HYDEPARK-Lösung

3.3 Einsatz in der HYDEPARK-Topologie

In dem Kapittel 3.1 ist gezeigt worden, daß es bei der Verwendung vom Wireless Application Protocol zu Problemen kommen kann, wenn WML-Dokumente eine vom Client oder Server verlangte maximale Übertragungskapazität überschreiten. Da HYDEPARK seine zu übermittelnden Daten nicht statisch bereitstellt, sondern erst zur Laufzeit generiert, muß in die Architektur ein Segmentierungsalgorithmus eingefügt werden. Die Struktur von HYDEPARK ist so ausgelegt, daß die ECAs die Kommunikation mit den Geräten vornimmt. Dabei ist der External Communictions Adapter als Servlet realisiert, so daß sich hier der ideale Punkt befindet, an dem ein dynamischer URL-Filter integriert werden kann[7]. Da es jedoch nur eine Implementierung des ECAs gibt, die sich den verschiedenen Protokollen anpaßt, darf sich das Filter-Element nur dann zuschalten, wenn WML-Dokumente übertragen werden.

Durch die Verwendung von JAVA-Servlets läßt sich die Integration der notwendigen Elemente ohne größere Veränderungen in der ursprünglichen Softwarekomponente vornehmen. Durch die Vererbungsmöglichkeiten der Programmiersprache JAVA kann die ECA Klasse leicht um die Funktionalität des URL-Filters, des Segmentierers und des Caches erweitert werden. Dies ist in der Abbildung 3.3 als Klassendiagramm graphisch dargestellt. Dabei wird der Filter direkt im erweiterten Servlet implementiert und die anderen Elemente in eigenen Klassen realisiert, deren Dienstleisung in Anspruch genommen wird. Durch den Zugriff auf die Methoden der Superklasse wird das zu segmentierende Dokument vom Gateway angefordert und in der Klassenhierarchie nach oben übergeben. Die erzeugten Segmente müssen für jeden Client in einem eigenen Cache bereitgehalten werden, damit die Zuordnung der Segmente gewährleistet bleibt.

[7] vergl. 3.2

Kapitel 4

Implementierung

4.1 Realisierung in JAVA

Die Implementierung greift alle bisher vorgestellten Elemente auf und führt diese zusammen. In 3.3 ist der ECA als idealer Platz zum Einsatz der Komponenten identifiziert worden. Um möglichst keine Veränderungen an den bestehenden Softwarekomponenten vornehmen zu müssen, wird von dem ECA-Servlet geerbt. Damit besizt das Segmentierer-Servlet alle Funktionen der Elternklasse, die noch um die des URL-Filters erweitert wird und die Dienste der `Segmentierer-` und der `PageCache`-Klasse in Anspruch nimmt. In der `service`-Methode, welche von dem Webserver aus automatisch aufgerufen wird, wird zunächst die `parseQuery`-Methode angesprochen, die angibt ob der Parameter 'page' überhaupt gesetzt ist und wenn ja, welchen Wert dieser hat. Ihr Kern besteht dabei aus einem `StringTokenizer`[1], der die Anfrage in ihre einzelnen Elemente aufbricht. Dabei wird zunächst der Parameter gesucht und falls vorhanden auf seinen Werte abgeprüft. Das Ergebnis wird als Rückgabewert der Methode gesetzt. Dieser Wert wird in der `service`-Methode weiterverarbeitet. Gibt es zu der Session, die zur Anfrage gehört, ein `PageCache`-Objekt im Verzeichnis der Seitenspeicher, so wird das entsprechende Objekt zur weiteren Verwendung herangezogen. Anderenfalls wird ein neues Objekt angelegt. Mit dem Ergebnis des Parameters und dem `PageCache`-Objekt wird entschieden, ob die angefragte Seite aus dem Speicher geholt werden kann. Sollte dies nicht der Fall sein, so wird eine neue Segmentierung angestoßen. Dabei wird die `service`-Methode der Elternklasse aufgerufen, die das Erstellen der Seite übernimmt. Diese darf jedoch nicht direkt in den Ausgabestrom des Servlets schreiben, sondern

[1] vergl. `java.util.StringTokenizer`, [5] Seite 567

bekommt ein eigenes `HttpServletResponse`-Objekt[2], welches einen modifizierten `PrintWriter`[3] als Ergebnis der Methode `getWriter` liefert. Mit der durch `MyHttpServletResponse` gekapselten Methode `getContent` kann dann auf den Inhalt des `PrintWriters` zugegriffen werden. Diese so erhaltene Seite wird dann an den `Segmentierer` übergeben und das Fragmentieren gestartet. Das Ergebnis wird über den ebenfalls übergebenen Seitenspeicher (`PageCache`) zur Verfügung gestellt. Daraus wird das erste Segment in den Ausgabestrom geschrieben.

Der `Segmentierer` bietet nur eine öffentliche Methode an. Dies ist die `segmentiere`-Methode, in der zunächst das als `String` übermittelte Dokument mit Hilfe der privaten Methode `separiereElemente` in ein Objekt des Types `Deck` überführt wird. Danach wird die Größe des `Deck`-Objekts ermittelt, welches sehr wohl von der des Originaldokuments abweichen kann, da überflüssige Whitespaces[4] elemeniert wurden. Aufgrund dieser Daten wird das weitere vorgehen entschieden. Ist das Dokument zu Groß, so wird sukzessive eine Card nach der anderen aus dem Deck entfernt und in einen eigenen eingefügt. Natürlich werden zuvor die Tags zur Navigation eingefügt. Sollten diese neuen Decks immer noch mehr Bytes belegen als erlaubt, so wird mit der `makeSegments`-Methode deren enthaltene Card fragmentiert. Diese Methode merkt sich alle noch offenen Tags in einem speziellem Kellerspeicher, dem `TagStack`. Dies ist notwendig, da durch die XML-Struktur eines WML-Dokuments alle geöffneten Tags auch in diesem wieder geschlossen werden müssen. Außerdem müßen im nächsten Dokument alle Tags des Stacks erneut geöffnet werden, da die Formatierung weiter gilt. Alle entstandenen Decks werden im `PageCache`-Objekt abgelegt und stehen so dem Servlet zur Verfügung.

Die `PageCache`-Klasse enthält als zentrales Objekt einen Vector[5] zur Aufnahme der Decks. Außerdem gibt es mehrere Methoden, um auf die einzelnen Elemente zugreiffen zu können.

4.2 Die `Segmentierer` Klasse

Im Mittelpunkt der Implementation steht die `Segmentierer`-Klasse. Sie segmentiert ein übergebenes WML Dokument in Teildokumente, die eine vorgegebene maximale Bytezahl nicht überschreiten. Nachdem eine Instanz von dieser Klasse erzeugt wurde, wird das Segmentieren durch den Aufruf der

[2] vergl. `javax.servlet.http.HttpServletResponse`
[3] vergl. `java.io.PrintWriter`, [5] Seite 447f
[4] Sonderzeichen, die lediglich zur Formatierung des Quelltextes dienen.
[5] vergl. `java.util.Vector`, [5] Seite 569f.

`segmentiere(String src, PageCache cache, int maxSize)`

Methode gestartet. Der Verlauf des Segmentierens folgt dabei den in Abbildung 4.1 und 4.2 dargestellten Flußdiagrammen. Die Ergebnisse werden in dem ebenfalls übergebenen `PageCache`-Objekt gespeichert und stehen so dem Aufrufenden zur Verfügung.

```
public class Segmentierer implements Serializable {

 protected static final int FIRSTCARD = 0;
 protected static final int MIDDLECARD = 1;
 protected static final int LASTCARD = 2;
 protected static final int NONAVIGATION = 3;

 protected Deck myDeckCache = new Deck();

 protected String _basis;

 protected CardRegister _register = new CardRegister();

 protected int _cardCounter = 0;

 protected SemanticCorrector _corrector;
```

Die Klasse beginnt mit der Deklaration der benötigten Variablen und Objekte. `FIRSTCARD`, `MIDDLECARD`, `LASTCARD` und `NONAVIGATION` sind Konstanten, die angeben, welche Navigationselemente in die Card eingefügt werden. Bei `FIRSTCARD` wird nur die Navigation eingefügrt, die zum nächsten Segment führt. Bei einer `MIDDLECARD` wird sowohl auf das nächste als auch auf das vorherige Segment verwiesen. Bei der `LASTCARD` wird nur der Link auf das vorherige Segment eingefügt und bei `NONAVIGATION` wird keine Navigation hinzugefügt. `myDeckCache` nimmt das in eine Objekt-Struktur überführte WML-Dokument auf, das segmentiert werden soll. Die URL des WML-Dokuments wird in dem `String _basis` gespeichert. Diese wird für das Einbinden der Navigation benötigt, wie auch die `_cardCounter`-Variable. In dem `CardRegister`-Objekts werden die neu geschaffenen Segmente mit dem id-Feld der entsprechenden Card des ursprünglichen Dokuments verknüpft. Dieses Verzeichnis wird dem `_corrector` übergeben, damit dieser die semantischen Korrekturen vornehmen kann.

Decksegmentierung

```
public void segmentiere(String src, PageCache cache, int maxSize)
throws SegmentToSmallException {

 separiereElemente(src);
 if ( myDeckCache.length()>maxSize) {
  [...]
 else {
   cache.addDeck(myDeckCache);
 }
}
```

Zunächst wird mit dem Aufruf der privaten Methode `separiereElemente` das WML-Dokument in eine Objektstruktur überführt. Auf dieses kann über das Objekt `myDeckCache` zugegriffen werden. Danach wird getestet, ob das Dokument überhaupt segmentiert werden muß. Sollte dies nicht der Fall sein, so wird der Deck ohne weitere Bearbeitung als einziges Dokument in den `PageCache` geschrieben. Anderen Falls beginnt das Segmentieren.

```
_register.clear();
_cardCounter=0;
[...]
setQuantity(maxSize);

if (myDeckCache.getHead()!=null){
 _corrector.correct(myDeckCache.getHead());
}
if (myDeckCache.getTemplate()!=null){
 _corrector.correct(myDeckCache.getTemplate());
}
 for (int j=0; j<myDeckCache.countCards(); j++){
 _corrector.correct(myDeckCache.cardAt(j));
}
```

Als Erstes werden die Variablen und Objekte auf die Anfangswerte zurückgesetzt und dann in der `setQuantity`-Methode die erwartete Anzahl von neuen Decks abgeschätzt und an den `SemanticCorrector` übergeben. Eine Abschätzung ist zunächst nötig, da die Anzahl der neu zu schaffenden Decks und die Größe der eingefügten semantischen Korrekturen von einander abhängen. Dabei wird es in kaufgenommen, nicht alle Bytes pro Segment

auszunutzen, da dies besser ist, als die maximale Anzahl zu überschreiten. Im Anschluß werden alle Elemente des Decks – also Head, Template und jede Cards – einer vorläufigen semantischen Korrektur unterzogen. Damit ist es möglich die Größe der Segmente annähernd bestimmen zukönnen.

```
while ((myDeckCache.countCards()>=1) &
(myDeckCache.length()>maxSize)) {

 Card card = myDeckCache.cardAt(0);
 myDeckCache.removeCardAt(0);

 makeSegments(cache,card,maxSize);

}
```

Durch die `while`-Schleife, in der überprüft wird, ob noch Cards im unsegmentierten Deck vorhanden sind und ob dieser weiterhin zu groß ist, wird die Decksegmentierung implementiert. Im Schleifenkörper wird bei jedem Durchlauf eine Card aus dem Deck entfernt und der Cardsegmentierung übergeben. Dies geschiet durch den Aufruf der geschützten `makeSegments`-Methode. Diese ist neben dem Segmentieren auch für das Einfügen der Cards in eigene Decks verantwortlich. Wenn keine Card mehr im Deck vorhanden ist oder der Restdeck die maximale Größe nicht mehr überschreitet ist die Decksegmentierung abgeschlossen.

```
if (myDeckCache.countCards()>0){
 cache.addDeck(myDeckCache);

 for(int j=0;j<myDeckCache.countCards();j++){
  Attribut[] attribute =
   myDeckCache.cardAt(j).getTag().getAttributs();
  String id="";
  for (int i=0; i<attribute.length;i++) {
   if (attribute[i].getName().equals("id")) {
    id=attribute[i].getValue();
   }
  }
  _register.insert(new RegisterEntry(id,_cardCounter
   (j==0)?true:false));
 }
}
```

Solte nach der Decksegmentierung ein Restdeck vorhanden sein, so wird dieser noch dem `PageCache`-Objekt hinzugefügt. Die im Restdeck enthaltenen Cards müssen dann noch im `CardRegister` eingetragen werden. Dabei wird bei jeder enthaltenen Card das id-Attribut gesucht und aus dessen Wert und der aktuellen Cardnummer (`_cardCounter`) ein neuer Registereintrag erzeugt.

```
for (int i=0; i<cache.pageCount();i++) {

 Deck deck = cache.deckAt(i);
 if (deck.getHead()!=null){
  _corrector.correct(deck.getHead());
 }
 if (deck.getTemplate()!=null){
  _corrector.correct(deck.getTemplate());
 }
 for (int j=0; j<deck.countCards(); j++){
  _corrector.correct(deck.cardAt(j));
 }
}
}
```

Als letztes wird in der `segmentiere`-Methode für jeden Deck die endgültige semantische Korrektur vorgenommen. Dabei werden in den Elemente, die einen Verweis enthalten, die Segmentnummern in den Links anhand der Einträge im Register angepaßt.

Cardsegmentierung

```
private void makeSegments(PageCache cache,Card card, int maxSize)
throws SegmentToSmallException {

 TagStack myTagStack = new TagStack();
 [..]
 boolean FirstSegment=true;

 while (card.hasMoreElements()) {
  [...]
 }
```

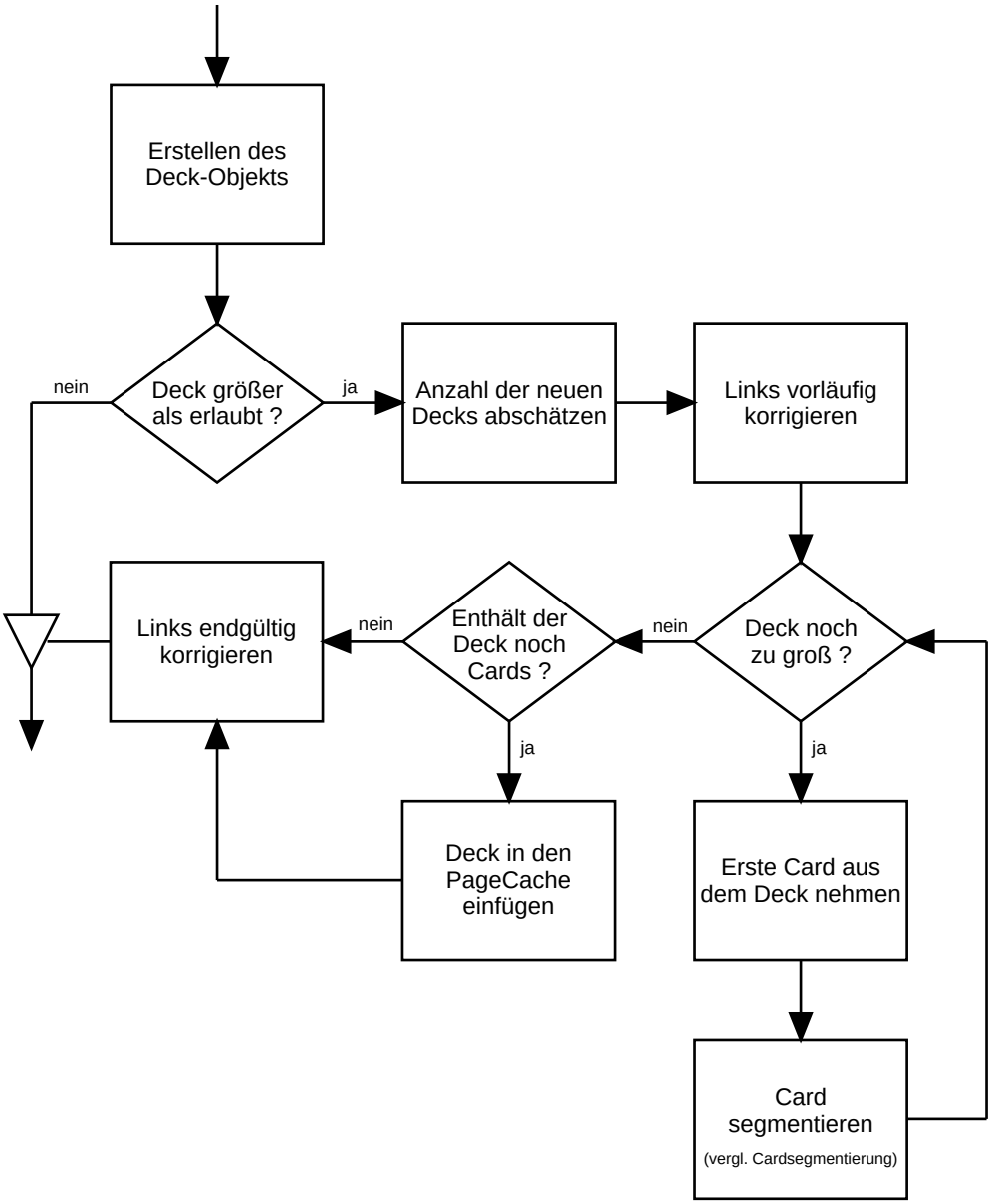

Abbildung 4.1: Flußdiagramm der Decksegmentierung

Die Methode `makeSegments` übernimmt die Aufgabe der Cardsegmentierung. Zunächst werden lokale Objekte und Variablen erzeugt. Dabei nimmt `myTagStack` WML-Tags auf, die für die Formatierung segmentübergreifend zur Verfügung stehen müssen. Die boolsche Variable `FirstSegment` wird gebraucht um erkennen zu können, ob beim Einfügen der Navigation mit dem Parameter `FIRSTCARD` oder `MIDDLECARD` gearbeitet werden soll. Durch die `while`-Schleife werden alle Elemente der Card auf neue Decks verteilt.

```
Card newCard = new Card(card.getTag());
Deck deck;
if(FirstSegment){
 deck = newDeck(null,_cardCounter++,FIRSTCARD);
}
else{
 deck = newDeck(null,_cardCounter++,MIDDLECARD);
}

int newMaxSize = maxSize-deck.length();

for (int i=myTagStack.size(); i>0; i--)
newCard.add(myTagStack.peekAt(i-1));
[...]
```

Zunächst wird eine neue Card und ein neuer Deck erzeugt. Dabei enthält die Card die gleichen Attribute wie das Original, sowie der Deck die Template- und Headelemente des ursprünglichen Decks. Dabei wird unterschieden, ob es sich um das erste Segment einer Card handelt oder um ein Folgesegment. Die Variable `newMaxSize` enthlt die Anzahl vn Bytes, die für die neue Card noch zur Verfügung stehen. Danach werden die Elemente, die sich auf dem Kellerspeicher (`myTagStack`) befinden in die Card eingefügt.

```
while (card.hasMoreElements() &
(newCard.length()+myTagStack.lengthOfCloseingTags())<=
newMaxSize & !exit) {
 Object element = card.nextElement();

 int elementLength = (element instanceof Tag)?
 ((Tag)element).length():((String)element).length();

 if(newCard.length()+myTagStack.lengthOfCloseingTags()+
 elementLength<=newMaxSize) {
```

```
    [...]
    if (element instanceof Tag) {
     Tag tag = (Tag)element;
     newCard.add(tag);

     if (!tag.isCloseingTag()) {
      myTagStack.push(tag);
     }
     else {
      if (!tag.isOpeningTag()) {
       myTagStack.pop();
      }
     }
    }

    if (element instanceof String) {
     newCard.add((String)element);
    }
   }
   else {
    [...]
    exit = true;
    card.resetLastElement();
   }
  }
```

In dieser Schleife wird nun das eigentliche kopieren der WML-Elemente vorgenommen. Zunächst wird das nächste Element aus der alten Card herausgenommen und seine Größe ermittelt. Wenn diese noch in die Card paßt,so wird es eingefügt. Dabei muß die kumulierte Anzahl der Bytes von dem Element, der Card und den noch zu schließenden Elemente auf dem Keller-Speicher kleiner oder gleich der maximalen Cardgröße sein. Sollte es sich um ein Tag handeln, so wird es, wenn es nur öffnend ist, zusätzlich noch auf den Keller gespeichert. Bei einem nur schließenden Tag, wird das zugehöhrende öffnende vom Keller genommen. Paßt das Element nicht mehr in die Card, so wird es zurückgesetzt und das Einfügen weiterer Elemente in diese Card abgebrochen.

```
  try {
   for (int i=0; i<myTagStack.size(); i++)
   newCard.add(Tag.valueOf(
```

```
 myTagStack.peekAt(i).getCloseingTag()));
}
catch (Exception e) {
 System.out.println(e);
}

deck.addCard(newCard);
cache.addDeck(deck);
if (FirstSegment) {
 FirstSegment=false;

 Attribut[] attribute = newCard.getTag().getAttributs();
 String id="";
 for (int i=0; i<attribute.length;i++) {
  if (attribute[i].getName().equals("id")) {
   id=attribute[i].getValue();
  }
 }
 _register.insert(new RegisterEntry(id,_cardCounter-1,true));
}
TagCorrector tagCorrector = new TagCorrector();
tagCorrector.removeEmptyElements(newCard);
}
```

Die Card ist noch nicht vollständig, da noch die fehlenden schließenden Tags eingefügt werden müssen. Dies geschieht in der ersten `for`-Schleife. Danach wird die Card in den Deck eingefügt und dieser in den `PageCache`. Durch setzten der `FirstSegment`-Variable wird angezeigt, daß nun nur noch Folgesegmente erzeugt werden. Die Card wird anschlißend ins Register eigetragen. Mit hilfe der `TagCorrector`-Klasse werden bei der Card alle Tags entfernt, die leer sind, dies aber laut Dokumenttyp-Beschreibung nicht sein dürfen.

```
Deck alterDeck = cache.deckAt(_cardCounter-1);
Deck neuerDeck ;
if (FirstSegment){

 neuerDeck= newDeck(null,_cardCounter-1,NONAVIGATION);
}
else{
 neuerDeck= newDeck(null,_cardCounter-1,LASTCARD);
```

Segment	Größe	Navigation
card1 / 1	892 Bytes	nächste
card1 / 2	840 Bytes	nächste, vorherige
card1 / 3	412 Bytes	vorherige
card2 / 1	253 Bytes	—

Tabelle 4.1: Größe der einzelenen Segmente

```
  }
  for (int i=0; i<alterDeck.countCards(); i++){
   neuerDeck.addCard(alterDeck.cardAt(i));
  }
  cache.setDeckAt(neuerDeck,_cardCounter-1);
 }
```

Nachdem alle Elemente der alten Card auf neue Segmente aufgeteilt sind, wird bei der letzten Card noch die Navigation angepaßt. Sollte es sich um eine einzelne Card handeln, so wird keine Navigation eingefügt. Beim letzten Segment einer Serie von Teilseiten dagegen, wird die Navigation nur auf die vorherige Seite gesetzt. Damit ist die Cardsegmentierung abgeschlossen.

4.3 Ein Beispiel

Die theoretischen Ausführungen über die Deck- und Cardsegmentation, sollen an dieser Stelle an einem Beispiel verdeutlicht werden. Als Grundlage dient dabei, die in Anhang D angegebene Pressemitteilung der Muster AG. Das Originaldokument hat eine Größe von 1621 Bytes. Die Segmentgröße darf 900 Bytes in diesem Beispiel nicht überschreiten.

Nachdem das Dokument den Segmentierer durchlaufen hat, sind insgesamt vier Segmente entstanden. Dabei ist die *card1* auf drei Cards verteilt und die *card2* ist komplett in einem Deck untergebracht. Die Größen der einzelnen Segmente kann aus Tabelle 4.3 entnommen werden.

Am Interessantesten ist das dritte Segment der *card1*, da hier sowohl eine Navigation eingefügt wurde als auch der enthaltene Link einer semantischen Korrektur unterzogen wurde.

```
<?xml version="1.0"?>
<!DOCTYPE wml PUBLIC "-//WAPFORUM//DTD WML 1.1//EN"
"http://www.wapforum.org/DTD/wml_1.1.xml">
```

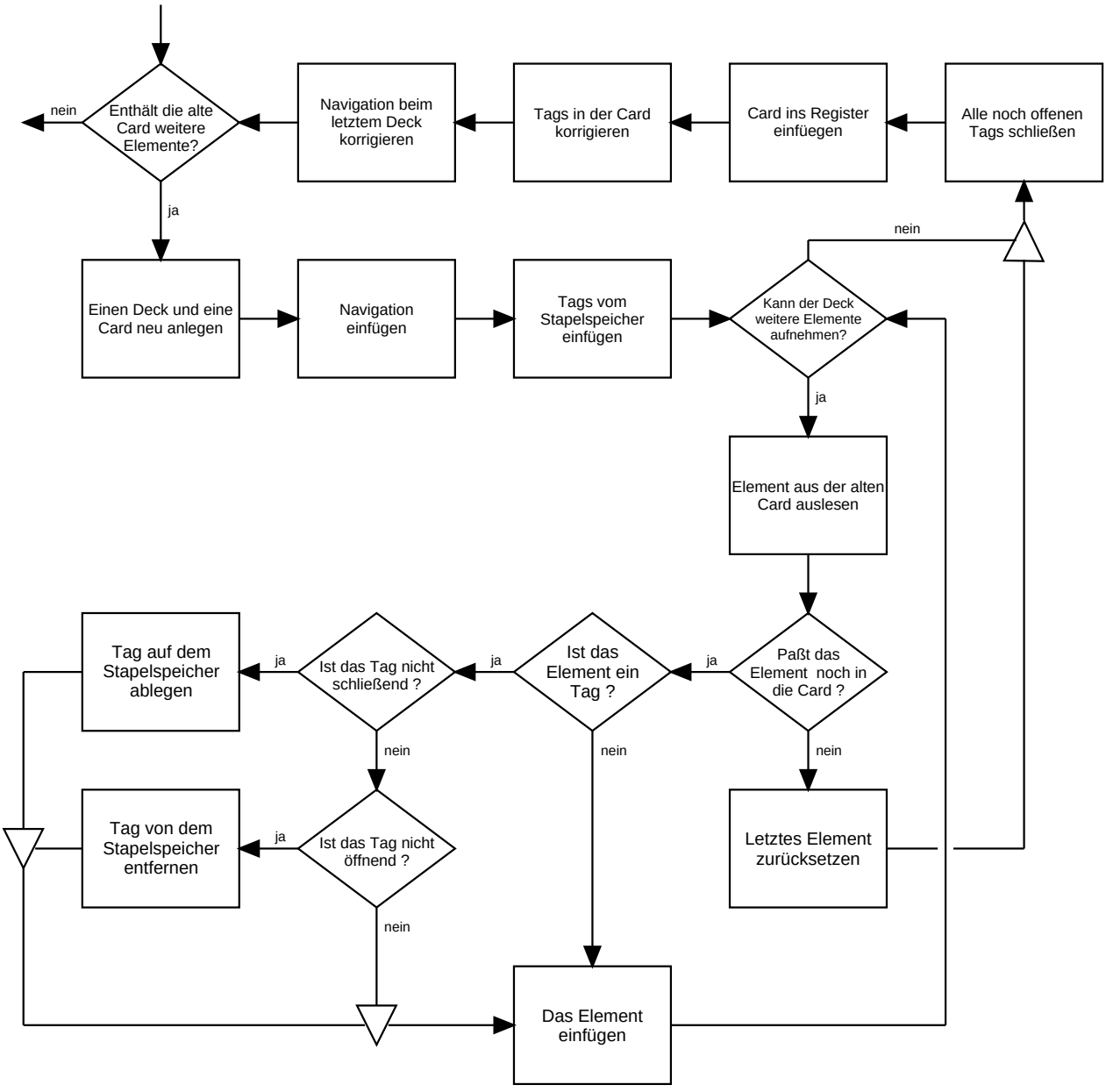

Abbildung 4.2: Flußdiagramm der Cardsegmentierung

Abbildung 4.3: Das dritte Segment

```
<wml>
 <template>
  <do name="PrSe" type="accept" label="Previous Segment">
   <go href="http://ford/servlet/SegmentWMLServlet?page=1"/>
  </do>
 </template>

 <card id="card1" title="PM Muster AG">
  <p>
   <a title="anzeigen"
   href="http://ford/servlet/SegmentWMLServlet?page=3#card2">
    Kontaktadresse
   </a>
  </p>
 </card>

</wml>
```

Da das ursprüngliche Dokument kein Template-Element enthalten hat, hat der Segmentierer diese hinzugefügt und die Navigation auf die vorhergehende Seite eingefügt. Bei dem `<a>`-Tag wurde das Attribut `href` der semantischen Korrektur unterzogen und verweist jetzt nicht mehr intern (`#card2`) sondern extern auf die Zielcard. Dieses WML-Segment ist in Abbildung 4.3 zu sehen, wie es sich auf einem Mobiltelefon darstellt.

Kapitel 5

Zusammenfassung und Ausblick

Das Internet über drahtlose Netze verfügbar machen, ist das Ziel, das das WAP-Forum mit dem Wireless Application Protocol verfolgt. Die im WAP-Protokollstapel enthaltenen Protokolle sind dabei an die Bedürfnisse der mobilen Kommunikation angepaßt. Um variabel mit der großen Zahl von Endgeräten umgehen zu können, besteht im Wireless Session Protocol die Möglichkeit, Fähigkeiten zwischen den Kommunikationspartnern abzustimmen. Unter anderem kann das Endgerät angeben, welche maximale Größe eine Dateneinheit, die an das Gerät gesendet wird, besitzen darf. Dies ist notwendig, da der zur Verfügung stehende Speicher der Geräte sehr unterschiedlich sein kann. Diese Einschränkung der Übertragungskapazität wirkt bis in die Anwendungsschicht des Protokollstapels hinein. Damit WML-Seiten auf dem Client angezeigt werden können, dürfen auch diese die maximale Datenpaketgröße nicht überschreiten. Damit ist es sehr schwer WML-Seiten zu gestalten, die auf möglichst allen Endgeräten angezeigt werden können, gerade wenn sich der Inhalt der Seiten dynamisch verändert. Eine Lösung dieses Problems ist der Einsatz eines Segmentierers, der eine gegebene Seite so in Fragmentseiten aufteilt, daß diese die jeweilige maximale Paketgröße nicht überschreiten.

Um eine Segmentierung von WML-Seiten durchzuführen, muß auf deren Struktur – die Einteilung in Decks und Cards – Rücksicht genommen werden. Es muß ebenfalls gültiger XML-Code erzeugt werden, so daß in bestimmten Fällen eine syntaktische Korrektur der Segmentseiten vorzunehmen ist. Aber auch semantische Anpassungen müssen in den neu geschaffenen Seiten vorgenommen werden, damit die Verweise zwischen den einzelen WML-Komponenten nach dem Segmentieren noch stimmen. Damit auf verschiedene Fragmente aufgeteilte WML-Elemente weiterhin in Beziehung stehen, muß an diesen Stellen eine Navigation zwischen den Sgementen eingefügt werden. Neben der Entwicklung der Theorie der Segmentierung von WML-Seiten,

steht die Implementierung für das HYDEPARK-Projekt.

Die Basis für die Entwicklung des WML-Segmentierers und die Implementierung für die JAVA Border Service Architecture des HYDEPARK-Projekts, ist die Darlegung der zu Grunde liegenden Technologiene im Kapitel 2. Im besonderen wird auf die im Wireless Sessicon Protocol definierten Fähigkeitsabsprachen eingegengen. Um den Lösungsansatz zu formulieren, ist es notwendig, sich über grundsätzliche Unterschiede der Segmentierung auf Anwendungsebene und der in anderen Protokollen Gedanken zu machen. Dies führt zur allgemeinen Segmentierung von WML-Seiten. Das Kapitel 3 schließt, mit Ausführungen, wie ein solcher Segmentierer in eine bestehende Umgebung im allgemeinen und in HYDEPARK im speziellen eingebunden werden kann. Die Umsetzung ist dann in JAVA realisiert.

Die Ergebnisse der Arbeit haben die JAVA Border Service Architecture so erweitert, daß die von ihr generierten WML-Seiten vor der Übertragung an den Client einer Segmentierung unterzogen werden können, um sie an das Endgerät anzupassen. Dabei kann die Segmentgröße individuell eingestellt werden. Dazu können Informationen aus dem Device Class Cataloge herangezogen werden. Die JBSA ist so in der Lage nicht nur auf verschiedene Sprachdialekte sondern auch auf technische Eigenschaften des Clients einzugehen. Da das Wireless Application Protocol eine noch relativ junge Technologie ist, bot sich mir durch diese Arbeit die Möglichkeit, selbst eine Lösung für das zu Grunde liegende Problem zu finden. Das Fehlen von umfangreicher Sekundärliteratur förderte das eigenständige Erarbeiten des Problems und damit ein fundiertes Wissen über WAP. Jedoch konzentriert sich die Arbeit nicht nur auf dieses Thema, sondern verlangt und förderte auch Kentnisse in anderen Technologien wie JAVA-Servlets, entfernter Methoden Aufruf (RMI) oder XML.

Der Segmentierer, wie er in dieser Arbeit entworfen wurde, kann noch weiter entwickelt werden. Eine Möglichkeit wäre ihn mit einem WML-Compiler zu kombinieren, der die WML-Seite in eine tokenbasierte Form überführt und so komprimiert. Der Segmentierer könnte dann auf der compilierten Repräsentation der WML-Seite arbeiten. Auch das Portieren aus der JBSA in andere Architekturen, wie dem WAP-Gateway oder einem Web-Server, bietet interessante Aspekte. Dabei könnte dann zum Beispiel der Mechanismus des Servlet Chaining eingesetzt werden.

Anhang A

Glossar

Cache: Ein Cache ist ein Pufferspeicher, der temporär benötigte Daten für einen schnellen Zugriff bereithält.

Client: Ein Gerät oder eine Anwendung, die eine Anfrage an einen → Server stellt. (vergl. [17] Seite 9)

Datagramm: In Zusammenhang mit Rechnernetzen wird damit eine Dateneinheit bezeichnet, die ohne vorherigen Verbindungsaufbau übertragen werden kann. (siehe [13] Seite 818)

Extensible Markup Language (XML): XML ist ein vom World Wide Web Consortium (W3C) definierter Standard für Auszeichnungssprachen für das Internet[...]. XML ist eine Untermenge von SGML.
(siehe [17] Seite 9)

Host: Hauptrechner (→ Server) in einem Rechnernetzwerk [...].
(siehe [3] Seite 199)

Internet Protocol (IP): Protokoll der Vermittlungsschicht im Internet.
(vergl. [13] Seite 52f)

Intranet: Firmeninternes Netzwerk, das auf den Internetprotokollen TCP/IP bzw. UDP/IP basiert.

JAVA: Plattformunabhängige, objektorientierte und statisch getypte Programmiersprache, deren Syntax auf C / C++ basiert.
(vergl. [9], The JAVA Platform)

JAVA SWING: Paket von Klassen zur Gestaltung von grafischen Benutzungsschnittstellen unter JAVA.

Link: Verweis zu anderen Dokumenten mittels einer → URL.

Netztopologie: Die Art und Weise, [...] wie die Endgeräte geographisch miteinander verknüpft sind. (siehe [7] Seite 135)

Overhead: Anteil eines Datenpakets, der keine Nutzdaten enthält.

Protokoll: Protokolle sind Regeln zur Kommunikation zwischen Partnerinstanzen. (siehe [7] Seite 37)

Schnittstelle: Allgemein die Verbindungsstelle zwischen Systemen, die zusammenarbeiten. (siehe [3] Seite 333f)

Server: Bezeichnung für einen Knotenrechner im Netzwerk, der seine Dienste allen anderen angeschlossenen Computern zur Verfügung stellt. (siehe [3] Seite 337)

Serviceprovider: Anbieter von Dienstleistungen in Mobilfunknetzen, die kein eigenes Netz besitzen.

Superklasse: Bezeichnung für die Elternklasse innerhalb der Erbhirarchie von JAVA-Klassen. Also ist die Superklasse die Klasse von der direct geerbt wird.

Tag: Ein Tag ist ein allgemeiner Term um ein Sprachelement zu beschreiben. WML [Dokumente] bestehen aus Inhalt, umgeben von formatierenden Tags. [...] Tags werden im allgemeinen in Paaren verwendet, eins um das Element beginnen zu lassen und eins um es zu beenden. (siehe [11] Seite 107)

Transaktion: Die Transaktion ist die Kommunikation zwischen Initiator einer Anfrage und dem Antwortenden. (vergl. [21] Seite 9 & [6] Seite 15)

Uniform Resource Locator (URL): Eine Spezifikation im World Wide Web, die die Syntax zur eindeutigen Adressierung von WWW-Dokumenten definiert. (siehe [13] Seite 831)

User Datagram Protocol (UDP): Unzuverlässiges → verbindungsloses Protokoll der Transportschicht[...]. (vergl. [13] Seite 831)

verbindungslos: Als verbindungslos wird eine Übertragung bezeichnet, wenn es keine taktische Phase des Verbindungsaufbaus und –abbaus gibt, sondern Daten ohne diese Vorbereitung gesendet und empfangen werden. (siehe [7] Seite 142)

Anhang B

WML-Elemente mit Verweisen

Hier sind alle WML-Tags zusammengefaßt, die einen Verweis enthalten. Die Attribute, die eine URL enthalten sind durch ein *href* (*hyperreference*) als Wertparameter gekennzeichnet.

- Decks und Cards

  <card id="*name*" title="*label*" newcontext="*boolean*"
  ordered="*boolean*" onenterforward="*href*"
  onenterbackward="*href*" ontimer="*href*"**>**

  Inhalt

  </card>

  <template onenterforward="*href*" onenterbackward="*href*"
  ontimer="*href*"**>**

  Inhalt

  </template>

- Tasks

  <go href="*href*" sendreferer="*boolean*" method="*method*"
  accept-charset="*charset*"

```
      Inhalt
</go>
```

- Benutzer Eingaben

```
<option title="label" value="value" onpick="href">
      Inhalt
</option>
```

- Anker

```
<a title="label" href="href">
      Text
</a>
```

Anhang C

WML-Elemente, die nicht leer sein dürfen

Hier sind alle WML-Tags zusammengefaßt, die in einem Dokument nicht leer enthalten sein dürfen.

- Events

```
<do type="type" label="label" name"name" optional="boolean">
      Aufgabe
</do>

<onevent type="type">
      Aufgabe
</onevent>
```

- Benutzer Eingaben

```
<select title="label" multiple="boolean" name="variable"
   value="value" iname="index_var" ivalue="default"
   tabindex="n">
      Inhalt
</select>

<optgroup title="label">
      Inhalt
</optgroup>
```

- Layout und Formatierung

```
<table title="value" align="alignment" columns="number">
      Inhalt
</table>

<tr>
</tr>
```

Anhang D

Das WML-Beispieldokument

```
<?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="card1" title="PM Muster AG">
  <p>
   <big>
    Muster AG auf Wachstumskurs
   </big>
   <br/>
   <b>
    Umsatzziel deutlich übertroffen
   </b>
   <br/>
   <br/>
   <i>Hamburg, 04. August 2000</i> - Die Muster AG, das
   f&#252;hrende IT-Beratungsunternehmen im deutschen
   Telekommunikationsmarkt, hat ein hervorragendes erstes Halbjahr
   hingelegt. Nach dem gelungenen Deb&#252;t am Neuen Markt der
   Frankfurter Wertpapierb&#246;rse am 19. Mai dieses Jahres sorgt
   Muster nun mit ihren aktuellen Unternehmenszahlen erneut
   f&#252;r gute Nachrichten.<br/>
   <br/>
   Das Halbjahresergebnis von Muster ist im Vergleich zu 1999
   deutlich h&#246;her ausgefallen. Bei den Umsatzerl&#246;sen
```

```
   sowie der Gesamtleistung verzeichnet der IT-Spezialist sogar
   hohe zweistellige Zuwachsraten. Derzeit w&#228;chst die Muster
   AG aus eigener Kraft um &#252;ber 50 Prozent in Deutschland.
   Damit liege man, so Finanzvorstand Max Meier, deutlich &#252;ber
   Plan und den zum B&#246;rsengang gemachten Ank&#252;ndigungen.
   Die genauen Zahlen werde das Unternehmen am 30. August bekannt
   geben.
   <br/>
   <br/>
   <a title="anzeigen" href="#card2">
    Kontaktadresse
   </a>
  </p>
 </card>
 <card id="card2" title=" Adresse ">
  <p align="center">
   <b>
    Muster AG
   </b>
   Pressestelle
   <br/>
   <em>
    20815
   </em>
   Hamburg
  </p>
 </card>
</wml>
```

Literaturverzeichnis

[1] AT&T Laboratories Cambridge. *Virtual Network Computing.* http://www.uk.research.att.com/vnc/.

[2] Au-System Radio. *WAP White Paper*, Februar 1999.

[3] Lexicon-Institut Bertelsmann. *Informatik, EDV, Computertechnik.* Bertelsmann Lexikon Verlag, 1994.

[4] Citrix Systems. *Independent Computing Architecture.* http://www.citrix.com.

[5] David Flannagan. JAVA IN A NUTSHELL, *Deutsche Ausgabe für JAVA 1.1.* O'Reilly, 2. edition, 1998.

[6] Rico Keller and Peter Angehrn. *Datenbankabfrage via Wireless Application Protocol.* Hochschule Rapperswill, Dezember 1999.

[7] Helmut Kerner. *Rechnernetze nach OSI.* Addison-Wesley, 3. edition, 1995.

[8] Rohit Khare. *W* Effect Considered Harmful.* 4K Associates, April 1999.

[9] Sun Microsystems. http://java.sun.com.

[10] Stefan Müller-Wilken. *JBSA: An Infrastructure for Seamless Mobile Systems Integration.* Universität Hamburg.

[11] Nokia Corporation. *WML Reference*, September 1999.

[12] swisscast. http://swisscast.ti-edu.ch.

[13] Andrew S. Tannenbaum. *Computernetzwerke.* Prentice Hall, 3. edition, 1996.

[14] W3C/MIT Network Working Group. *Hypertext Transfer Protocol – HTTP/1.1*, Juni 1999.

[15] WAP-Forum. *Wireless Application Protocol Architecture Specification*, April 1998.

[16] WAP-Forum. *WAP White Paper: Wireless Application Protocol*, Oktober 1999.

[17] WAP-Forum. *Wireless Application Protocol Wireless Markup Language*, November 1999.

[18] WAP-Forum. *Wireless Datagram Protocol Specification*, November 1999.

[19] WAP-Forum. *Wireless Session Protocol Specification*, November 1999.

[20] WAP-Forum. *Wireless Application Protocol Wireless Application Environment Overview*, März 2000.

[21] WAP-Forum. *Wireless Transaction Protocol Specification*, Februar 2000.

Warenzeichen / Trademarks

Alle in dieser Studienarbeit verwendeten Produktnamen sind Warenzeichen der rechtmässigen Eigentümer.

All product names mentioned herein are the trademarks of their respective owners.