Zugang zu Unternehmensinformation mit mobilen Kommunikationsgeräten

Diplomarbeit

an der

Technischen Universität Graz

vorgelegt von

Markus Wiederkehr

Begutachter: Univ.-Doz. Dipl.-Ing. Dr.techn. Frank Kappe

Institut für Informationsverarbeitung und Computergestützte Neue Medien (IICM)
Technische Universität Graz

Zusammenfassung

Die vorliegende Diplomarbeit zeigt am Beispiel des Hyperwave IS/6, wie der drahtlose Zugang zu einem Document Management System realisiert werden kann. Damit ist gemeint, dass zusätzlich zur gewohnten Zugangsmöglichkeit mit einem Personal Computer ein Personal Digital Assistent oder ein Mobiltelefon verwendet werden kann, um auf die im System abgelegten Dokumente zuzugreifen.

Der Hyperwave Informationen Server gestattet die Entwicklung und Ausführung von Applikationen, die über das reine Document Management hinausgehen. Eine dieser Applikationen, das Hyperwave eKnowledge Portal, ist ebenfalls Thema dieser Arbeit. Es wird beschrieben, wie dieses Portal dahingehend erweitert werden kann, dass seine Komponenten, die so genannten Tracks, so programmiert werden können, dass ihnen der drahtlose Zugang ermöglicht wird.

Als technologische Grundlage für die praktische Umsetzung wurde das *Wireless Application Protocol* (WAP) und das *Wireless Application Environment* (WAE) verwendet. Der erste Teil der Arbeit bietet einen Überblick über diese Bereiche. Hier werden die bei WAP verwendeten Protokolle und die Infrastruktur ebenso beschrieben wie die zum Einsatz kommenden Sprachen *Wireless Markup Language* (WML) und *WMLScript*. Außerdem wird noch auf das *Wireless Telephony Application Interface* (WTAI) eingegangen, eine Erweiterung von WAP, die die Erstellung von Anwendungen für die Telefonie ermöglicht.

Der zweite Teil der Arbeit widmet sich der Realisierung des drahtlosen Zugangs zum Hyperwave IS/6. Es wird die Erstellung der WAP-Komponente beschrieben, die grundlegende Aufgaben übernimmt, zum Beispiel die Anmeldung des Benutzers und die Ausgabe von Hyperwave-Objekten (wie etwa Collections oder gespeicherte Dokumenten). Darauf aufbauend wird dann die Modifikation des Portals erklärt, die die Voraussetzung für die Erstellung WAP-fähiger Tracks schafft.

Die Implementierung der WAP-Komponente und Anpassung des Portals wird mit Beispielen in JavaScript illustriert. Dabei werden aufgetretene Probleme und deren Lösungen betrachtet.

Danksagung

Ich danke Dr. Frank Kappe, auf dessen Anregung hin diese Diplomarbeit entstanden ist. Er hat nicht nur die Betreuung der Arbeit übernommen, sondern mir auch die interessante Tätigkeit bei Hyperwave ermöglicht.

Da diese Arbeit am Ende meines Studiums steht, möchte ich mich auch bei all jenen bedanken, die mir während der letzten Jahre und speziell während der Zeit der Diplomarbeit zur Seite standen.

Dazu gehören viele Freunde, die hier nicht alle namentlich erwähnt werden können. Stellvertretend danke ich Sandra Dominikus, Bernhard Ibertsberger, Christian Köbrunner, Stefan Pree, Hannes Schachtner, Rudolf Schamberger, Anton Schmidbauer und Till Vollmer für ihre Unterstützung beim Studium und/oder im Privaten und den vielen Spaß den wir gemeinsam hatten. Sie haben wesentlich dazu beigetragen, dass mir das Studium in guter Erinnerung bleiben wird.

Meiner Freundin Sandra Frühwirth danke ich für ihre moralische Unterstützung, ihre oftmalige Aufmunterung und für das Korrekturlesen dieser Arbeit.

Ganz besonderer Dank gilt schließlich meinen Eltern für das Ermöglichen dieser Ausbildung und für ihre Unterstützung und Ermutigung während dieser Zeit.

Inhaltsverzeichnis

Einleitung

Das Thema dieser Arbeit ist die Realisierung des drahtlosen Zugangs zum Hyperwave IS/6. Der IS/6 ist eine von der Firma Hyperwave entwickelte Plattform, die die Grundvoraussetzungen für Anwendungen im Wissensmanagement- und Web Based Training-Bereich schafft. Die von Hyperwave selbst entwickelten Anwendungen sind im Einzelnen:

Die Hyperwave eKnowledge Suite (EKS) ist eine Lösung für unternehmensweites Wissensmanagement, die die Ablage und Abfrage von strukturierten und unstrukturierten Daten ermöglicht. Für diese Tätigkeiten und für die Administration des Systems kann ein Web-Browser verwendet werden.

Das Hyperwave eKnowledge Portal (EKP) ist eine Portal-Lösung, die besonders für Teamarbeit konzipiert ist. Die Benutzer können sich ihren persönlichen Desktop nach individuellen Bedürfnissen aus so genannten Portal Tracks (das sind z. B. Komponenten zur Darstellung von E-Mail-Nachrichten, Aktienkursen, aktuellen Nachrichten u. v. m.) zusammensetzen. Hyperwave bietet Entwicklern eine offene Architektur zur Erstellung solcher Tracks an.

Die Hyperwave eLearning Suite (ELS) ermöglicht Web Based Training, also das Lernen im Intra- bzw. Internet. Die ELS kann mit dem Wissensmanagement-System EKS kombiniert werden.

Diese Arbeit zeigt, wie auf die beiden ersten dieser Anwendungen, die Hyperwave eKnowledge Suite und das Hyperwave eKnowledge Portal, auf drahtlosem Wege, mit einem WAP-fähigen Mobiltelefon oder einem anderen WML-Browser, zugegriffen werden kann.

Zur Realisierung des drahtlosen Zugangs wurde das Wireless Application Protocol (WAP) verwendet. Der erste Teil dieser Arbeit beschreibt WAP, um die Voraussetzungen für das Verständnis des praktischen Teils zu schaffen. Es werden die Architektur von WAP und im Speziellen das Wireless Application Environment (WAE)

erklärt. Zum WAE gehören die Wireless Markup Language (WML), das ist eine Sprache zur Beschreibung von Hypertext und das Äquivalent zu HTML im drahtlosen Bereich, die Programmiersprache WMLScript, die das Gegenstück zu JavaScript bildet, und das Bildformat WBMP als Ersatz für im World Wide Web gebräuchliche Formate wie GIF oder JPEG.

Der zweite Teil der vorliegenden Arbeit wendet sich der praktischen Herstellung des drahtlosen Zugangs zu den oben angeführten Hyperwave-Anwendungen zu. Dieser wird auf die Weise hergestellt, dass der Benutzer dieselben Daten und dieselbe Hierarchie wie im Web-Browser vorfindet. Daraus ergeben sich für ihn folgende wesentliche Vorteile:

- Der Anwender kann sich wie gewohnt orientieren, da ihm die Namen der Verzeichnisse und Dokumente sowie ihr Ort vom Web-Zugang her bekannt sind. Die hierarchische Struktur der Daten bleibt dieselbe.

- Bereits am Server existierende Dokumente können mit dem WML-Browser betrachtet werden, soweit dies angesichts der technischen Limitierungen dieser Geräte möglich ist.

- Der Anwender erhält Zugriff auf seinen persönlichen Desktop. Das Portal und die Tracks werden also ebenfalls auf drahtlosem Wege zur Verfügung gestellt.

Der zentrale Punkt dabei ist, dass der Benutzer bereits mit dem System vertraut ist und keine zusätzlichen Konfigurationen für den drahtlosen Zugriff erforderlich ist. Er muss sich also beispielsweise keinen eigenen Desktop für den drahtlosen Zugang einrichten, sondern ihm wird ohne besonderen Aufwand sein persönlicher Desktop auch am Mobiltelefon zur Verfügung gestellt.

Diese Arbeit beschreibt, welche Modifikationen in den Hyperwave-Anwendungen vorgenommen werden müssen, um die beschriebenen Anforderungen erfüllen zu können. Dabei werden zunächst grundlegende Punkte, wie die Identifizierung eines WAP-Clients durch den Server oder die Anmeldung des Benutzers, behandelt. Als Nächstes wird beschrieben, wie mit dem WML-Browser durch die hierarchische Struktur navigiert werden kann. Dazu gehört die Ausgabe des Inhalts von Verzeichnissen (bzw. Collections in der Hyperwave-Terminologie) und von Dokumenten. Im Speziellen wird die Ausgabe von Text- und HTML-Dokumenten sowie WBMP-Bildern beschrieben.

Bei all diesen Ausgaben spielt die begrenzte Speicherkapazität von WML-Browsern eine große Rolle. Die Arbeit beschäftigt sich auch damit, wie dieser Speicherplatz bestmöglich genützt werden kann. Da die Daten, die vom Server erzeugt und zum Client transportiert werden, üblicherweise durch ein so genanntes WAP-Gateway komprimiert werden, ist die optimale Speicherplatznutzung nicht auf einfache Weise zu gewährleisten. Als Lösung für dieses Problem wird ein Algorithmus zur effizienten Schätzung der Größe der komprimierten Daten vorgestellt.

Diese Anpassungen des Systems bilden in ihrer Gesamtheit die sogenannte WAP-Komponente. Die Arbeit erklärt, wie diese Komponente in die Templates der Hyperwave eKnowledge Suite integriert wird und wie dabei bestehende Konzepte zur Modularisierung des Systems berücksichtigt und verwendet werden.

Darauf aufbauend wird mit der Erweiterung des Portals fortgesetzt. Es werden die Rahmenbedingungen geschaffen, die es ermöglichen, Tracks zu erstellen, die sowohl über Web- als auch über WML-Browser nutzbar sind. Das bedeutet nicht, dass jeder Track ohne Anpassungen mit einem Mobilfunkgerät verwendbar ist, jedoch wird den Track-Entwicklern die Möglichkeit geboten, auf einfache Weise WML-Unterstützung zu ihren Tracks hinzuzufügen. Dem Anwender werden dann automatisch nur jene Tracks seines Desktops am Mobiltelefon bzw. WML-Browser präsentiert, die dafür auch ausgelegt sind.

Die Beschreibung der Anpassungen des Portals beginnt damit, wie der Anwender zu einem einzelnen Element bzw. Track eines Desktops gelangen kann. Das unterscheidet sich vom Web-Zugang, da der Desktop auf der vergleichsweise kleinen Anzeige eines Mobiltelefons nicht zur Gänze darstellbar ist. Dann wird beschrieben, wie die Ausgabe eines Tracks zu erfolgen hat und welche Erleichterungen dem Track-Entwickler durch das Framework geboten werden. Dazu gehört auch die Ausgabe von Hilfetexten ohne programmiertechnischen Aufwand. Zuletzt wird das Portal-Framework anhand eines praktischen Beispiels illustriert: Es wird gezeigt, wie ein Track erstellt werden kann, der auf ein POP3-E-Mail-Konto zugreift und dem Benutzer das Lesen dieser Nachrichten ermöglicht.

Teil I.

Das Wireless Application Protocol (WAP) und das Wireless Application Environment (WAE)

1. Übersicht über WAP

Der Begriff WAP steht für *Wireless Application Protocol*. WAP ermöglicht die Entwicklung von Applikationen und Diensten, die über ein drahtloses Netzwerk erreichbar sind. Die Geräte, auf denen die Applikationen laufen, können Mobiltelefone, Pager oder auch Personal Digital Assistents (PDAs) sein.

WAP spezifiziert ein Application Framework und die notwendigen Netzwerkprotokolle für die Übermittlung von Daten aus dem Internet zu einem mobilen Kommunikationsgerät bzw. umgekehrt. Die Übertragung erfolgt dabei teilweise im Internet und teilweise auf drahtlosem Weg. Im Gegensatz zum *World Wide Web*, bei dem sowohl Client als auch Server Bestandteil des Internets sind und die gleichen Protokolle sprechen und verstehen, muss bei WAP ein Gateway[1] existieren, welches die Inhalte und Protokolle auf die Bedürfnisse und Gegebenheiten des jeweiligen Mediums anpasst bzw. übersetzt.

Das WAP-Gateway macht es möglich, Inhalte auf herkömmlichen WWW-Servern abzulegen. Auch bewährte Technologien wie CGI-Scripts, Java-Servlets und dergleichen können zum Einsatz kommen, um dynamische Inhalte zu generieren.

Bei der Übersetzung werden beispielsweise die vergleichsweise geringen Bandbreiten und größeren Verzögerungen der drahtlosen Netzwerke berücksichtigt, indem die Daten aus dem Internet in eine kompakte binäre Darstellungsform gewandelt werden. Auf dem umgekehrten Weg werden die Daten wieder in die im Internet übliche Textrepräsentation gebracht.

Nicht nur die drahtlosen Kommunikationsnetzwerke hinken in ihrer Leistungsfähigkeit dem Internet hinterher: Die mobilen Kommunikationsgeräte selbst haben im Vergleich zu den heute üblichen Personal Computern wesentlich weniger Arbeitsspeicher, langsamere Prozessoren und sehr viel kleinere Displays, was die Darstellung von HTML-Seiten auf diesen Geräten zurzeit schwierig macht.

Ein Teil der WAP-Spezifikationen, das *Wireless Application Environment* (WAE), definiert daher schlankere Äquivalente zu den im World Wide Web üblichen Sprachen

[1] wird auch als *WAP-Gateway* oder *WAP-Proxy* bezeichnet.

HTML und JavaScript. Die mit HTML vergleichbare Sprache heißt WML (*Wireless Markup Language*), JavaScript findet seine Entsprechung in WMLScript. Für beide Sprachen gibt es eine binäre Darstellung und eine Textrepräsentation; das oben angesprochene Gateway übernimmt die Übersetzung jeweils von der einen in die andere Form.

Die im World Wide Web gebräuchlichsten Bildformate (hauptsächlich JPEG und GIF) sind für mobile Kommunikationsgeräte auf Grund ihrer Größe und Farbtiefe ebenfalls schlecht geeignet. Daher stellt WAP für Bilder ein eigenes Format mit der Bezeichnung WBMP (*Wireless Bitmap*) zur Verfügung.

Zu diesen Sprachen und Formaten, die ihre direkte Entsprechung im WWW haben, kommen noch andere Datenformate, wie zum Beispiel Telefonbucheinträge und Kalenderinformationen, hinzu. Außerdem gibt es noch telefoniespezifische Dienste und Programmierschnittstellen, die unter den Begriffen WTA (*Wireless Telephony Application*) und WTAI (*Wireless Telephony Application Interface*) zusammengefasst werden.

Folgende Anwendungen werden durch WAP ermöglicht:

- Abruf von Nachrichten
- Ortsbezogene Dienstleistungen, wie z. B. Echtzeit-Verkehrsberichte, Informationen über aktuelle lokale Ereignisse, Restaurantempfehlungen und dgl.
- Unternehmenslösungen wie etwa E-Mail-Abfrage, Datenbankzugriffe oder globaler Internetzugang
- Finanzielle Dienstleistungen; Überweisungen, Aktienhandel, ...
- Dienste für unterwegs, wie zum Beispiel die Abfrage von Fahrplänen oder Fahrplanänderungen, die Reservierung von Verkehrsmitteln oder Unterkünften usw.
- Spiele und Unterhaltung: Echtzeitspiele für mehrere Spieler, downloadbare Horoskope, Witze, Cartoons und vieles mehr
- mobiler Handel (*M-Commerce*), Preisvergleich, Abfrage örtlicher Sonderangebote usw.

1.1. Die WAP-Architektur

Dieser Abschnitt gibt eine Übersicht über die WAP-Architektur. Die Spezifikationen für WAP 1.2 sind unter der URL http://www.wapforum.org/ verfügbar. Dort finden sich auch noch die Spezifikationen von WAP 1.1, auf die hier allerdings nicht näher eingegangen wird.

Die WAP-Architektur bietet eine skalierbare und erweiterbare Umgebung für die Entwicklung von Applikationen für mobile Kommunikationsgeräte. Die Protokolle lassen sich in einem Schichtenmodell darstellen, wobei Protokolle in unteren Schichten von allen darüber liegenden Schichten verwendet werden können (siehe Abb. 1.1). Anders gesagt: Protokolle in höheren Schichten bauen auf die in darunter liegenden auf. Das Modell wird also von unten nach oben zunehmend abstrakter und entfernt sich mehr und mehr von der Notwendigkeit, physikalische Gegebenheiten berücksichtigen zu müssen. Die WAP-Protokolle und ihre Funktionen ähneln also dem ISO OSI Referenzmodell (vgl. [1]).

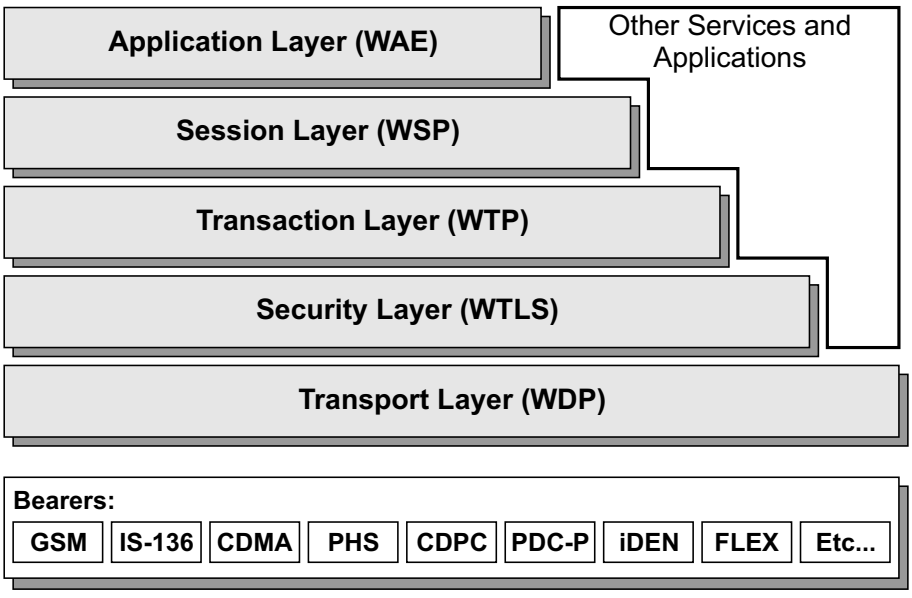

Abbildung 1.1.: Die WAP-Architektur (aus [2, Figure 4])

Jede Schicht verfügt über ein wohldefiniertes Interface, welches außer den höheren Schichten auch noch externen Diensten bzw. Anwendungen Zugriffe erlaubt. Eine externe Anwendung kann daher beispielsweise den Session Layer direkt verwenden, ohne den obersten Application Layer kennen oder verwenden zu müssen.

Wie in Kapitel 1 bereits angesprochen wurde, erfolgt die Kommunikation zwischen Server und Client über ein WAP-Gateway. Der Server befindet sich dabei im Internet, während der Client ein mobiles Kommunikationsgerät ist, welches nur auf drahtlosem Wege zu erreichen ist. Dazwischen befindet sich das Gateway, welches Zugang zu beiden Bereichen hat und die nötigen Umwandlungen der Protokolle und Inhalte vornimmt. Die einzelnen Schichten der WAP-Architektur beziehen sich größtenteils auf die Kommunikation des Gateways mit dem Client. Die Kommunikation zwischen Gateway und Server erfolgt wie im World Wide Web über das *Hypertext Transfer Protocol* (HTTP) und TCP/IP. Eine Ausnahme davon bildet die oberste Schicht der Architektur — hier sind die Sprachen WML und WMLScript spezifiziert, die auch zwischen Gateway und Server ausgetauscht werden, wenngleich normalerweise in ihrer Textrepräsentation.

Die einzelnen Schichten der WAP-Architektur, die in ihrer Gesamtheit den so genannten *WAP Protocol Stack* bilden, werden zunächst der Reihe nach kurz vorgestellt:

Application Layer: Die dieser Schicht zugeordnet Protokollfamilie ist das *Wireless Application Environment* (WAE). Im Application Layer werden das *World Wide Web* (WWW) und Technologien der Mobiltelefonie kombiniert. Dadurch entsteht ein flexibles Werkzeug zur Erstellung von Inhalten für mobile Endgeräte. Die wohl wichtigste Komponente des WAE ist die *Wireless Markup Language* (WML) — eine HTML-ähnliche Markup-Sprache, basierend auf XML.

Session Layer: In dieser Schicht der WAP-Architektur befindet sich das *Wireless Session Protocol* (WSP). Das WSP stellt die Mittel zur Verfügung, um Inhalte zwischen Client und Server austauschen zu können. Es bedient sich dazu der unteren Schichten WTP und WDP. Bei Verwendung der Transaktionen von WTP kann eine verbindungsorientierte Sitzung aufgebaut werden. Werden statt derer die Datagramm-Dienste des WDP verwendet, ist die Sitzungsart verbindungslos. Das WSP ermöglicht die Unterbrechung und Wiederaufnahme der Sitzung, was im drahtlosen Bereich wegen vieler unvorhersehbarer Unterbrechungen wichtig ist. Im Wesentlichen ist WSP eine binäre Form von HTTP/1.1 (*Hypertext Transfer Protocol*, siehe [3]). Das Wireless Session Protocol ist in [4] spezifiziert.

Transaction Layer: An dieser Stelle befindet sich das *Wireless Transaction Protocol* (WTP). Es sorgt dafür, dass Transaktionen zwischen Client und Server zuverlässig ablaufen. Unter einer Transaktion ist in diesem Zusammenhang die Einheit aus Anfrage des Clients und Antwort des Servers zu verstehen. Das WTP bietet den oberen Schichten des Protocol Stacks drei Klassen von Übertragungszuverlässigkeit, um den unterschiedlichen Bedürfnissen der Anwendungen gerecht zu werden. Die Spezifikation von WTP findet sich unter [5].

Security Layer: Das Protokoll dieser Schicht, das *Wireless Transport Layer Security* (WTLS), stellt die Mittel für eine sichere Übertragung zur Verfügung. Diese Schicht der WAP-Architektur ist optional, das heißt, es ist Sache des Entwicklers, zu entscheiden, ob eine sichere Übertragung notwendig ist oder nicht. Mit Sicherheit ist gemeint, dass WTLS Datenschutz, Datenintegrität und Authentifizierung zwischen zwei Applikationen gewährleistet. WTLS ist abgeleitet von TLS (siehe [6]), das wiederum auf den SSL 3.0 (*Secure Sockets Layer*) Spezifikationen basiert. Für genauere Informationen zu WTLS siehe [7].

Transport Layer: Diese Schicht vermittelt zwischen den oberen Schichten der Architektur und den verschiedenen Bearer Services (*Überbringer-* bzw. *Träger*-diensten) der untersten Schicht. Sie stellt den oberen Schichten ein einheitliches Interface zur Verfügung und befreit sie dadurch von der Notwendigkeit, die verschiedenen Bearer Services kennen zu müssen bzw. auf ihre unterschiedlichen Bedürfnissen eingehen zu müssen. Das dieser Schicht zugehörige Protokoll ist das *Wireless Datagram Protocol* (WDP). Die Spezifikation von WDP besagt, dass das *User Datagram Protocol* (UDP) verwendet werden muss, wenn der verwendete Trägerdienst das *Internet Protocol* (IP) unterstützt. Dies ist beim Einsatz von GSM Circuit-Switched Data, GSM GPRS oder einer besseren Technologie der Fall. Weiter führende Informationen zu WDP finden sich in [8].

Bearers: Die WAP-Protokolle sind in der Lage, mit den unterschiedlichsten Bearer Services zusammenzuarbeiten. Diese Trägerdienste sind in der untersten Schicht der WAP-Architektur angesiedelt. Beispielsweise wird so einen Übertragung über GSM SMS (*Short Message Service*) oder GSM USSD (*Unstructured Supplementary Service Data*) möglich.

Für diese Arbeit ist hauptsächlich die oberste Schicht, also der Application Layer, von Interesse. Sie wird im folgenden Abschnitt ausführlicher beschrieben.

1.2. Das Wireless Application Environment (WAE)

An oberster Stelle des WAP Protocol Stacks befindet sich das *Wireless Application Environment* (WAE), also die Umgebung, in der drahtlose Anwendungen laufen können.

Das WAE kann als eine Menge von Spezifikationen angesehen werden, die Sprachen, Datenformate und Bibliotheken abdecken, die von den Anwendungen benötigt werden bzw. auf deren Grundlage die Anwendungen überhaupt erst erstellt werden können. Dazu gehören die Markup-Sprache WML, die Skriptsprache WMLScript mit ihren Standardbibliotheken, das Bildformat WBMP und Spezifikationen für Telefonieanwendungen (WTA und WTAI).

Die Idee hinter WAE wurde vom Internet und speziell dem World Wide Web inspiriert. Daher ist auch das verwendete Modell ein ähnliches. Im WWW gibt es Server, welche Inhalte bereitstellen oder generieren, die von Clients mit einem Web Browser abgefragt werden können. Der Client ist dabei der aktive Part — er sendet eine Anfrage in Form einer URL (*Uniform Resource Locator*) an den Server, der den damit assoziierten Inhalt (oft als HTML-Seite) an den Client zurückschickt.

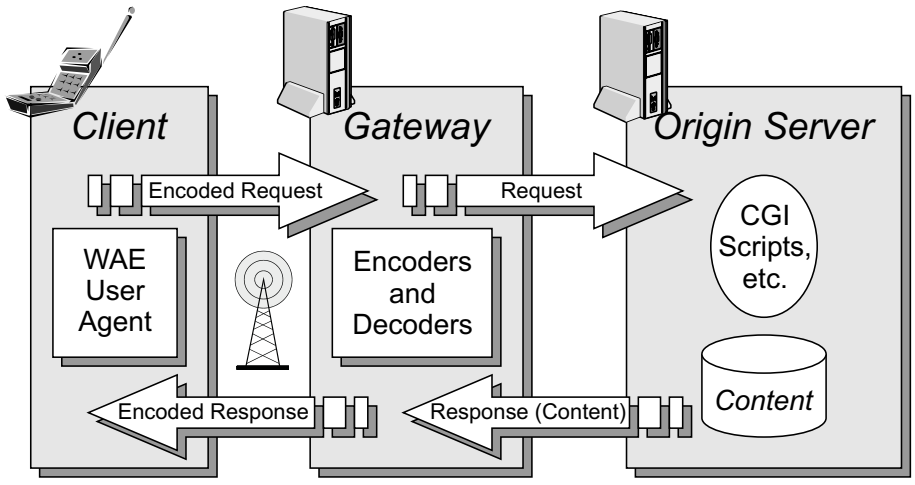

Abbildung 1.2.: Das WAE-Modell (aus [9, Figure 2])

Das WAE-Modell ist daran angelehnt, wie die Abbildung 1.2 auf der vorherigen Seite zeigt. Der Client sendet ebenfalls eine Anfrage an den Server, die dieser beantwortet.[2] Der wesentliche Unterschied zum WWW besteht darin, dass zwischen Server und Client das bereits erwähnte Gateway steht. Der Server befindet sich wie im WWW-Modell im (verdrahteten) Internet und seine Inhalte werden ebenfalls über URLs referenziert.

WML- und WMLScript-Dokumente können am Server in Klartextform vorliegen und werden dann vom Gateway komprimiert bzw. kompiliert, bevor sie an den Client geschickt werden. Dies beschleunigt einerseits die Übertragung, da die Bandbreite im drahtlosen Bereich wesentlich geringer ist als im Internet; andererseits wird auch der Rechenaufwand für den Client bei der Darstellung oder Ausführung verringert. Das Gateway vermittelt außerdem zwischen HTTP und seinem kompakteren Äquivalent im drahtlosen Bereich, WSP. Das WAP-Gateway fungiert also als Vermittlungs- und Schnittstelle zwischen dem drahtlosen Bereich und dem World Wide Web.

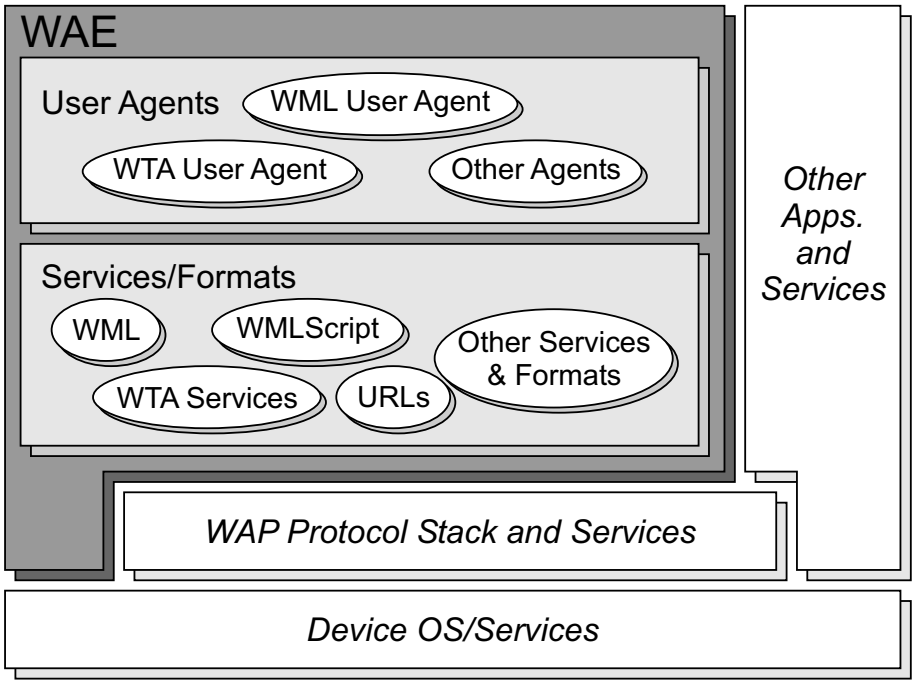

Abbildung 1.3.: Die WAE Client Komponenten (aus [9, Figure 4])

[2]Die Ausnahme dazu ist die *Wireless Telephony Application*, bei der Services von einem WTA-Server am Client installiert werden können, ohne dass dies vom Client initiiert wird.

Die Abbildung 1.3 auf der vorherigen Seite zeigt die Komponenten eines WAE-Clients im Kontext mit anderen WAP-Bestandteilen. Der fundamentale User Agent von WAE ist der *WML User Agent*. Er ist in der Lage, kodierte WML-Seiten darzustellen und vorkompiliertes WMLScript auszuführen. Wenn der Client ein Mobiltelefon ist, kann darüber hinaus auch noch ein *WAE User Agent* existieren, mit dem sich Features des Telefons, wie zum Beispiel das eingebaute Adressbuch, nutzen lassen.

Die angesprochenen Sprachen und Formate WML, WMLScript, WBMP, WTA und WTAI sind das Thema der folgenden Kapitel.

2. Die Wireless Markup Language (WML)

WML ist eine Markup-Sprache, welche auf XML (*Extensible Markup Language*) basiert — jedes WML-Dokument ist also auch ein gültiges XML-Dokument.[1] WML wurde speziell für drahtlose Geräte mit niedriger Bandbreite, wie zum Beispiel Mobiltelefone, Pager oder PDAs (*Personal Digital Assistents*), entwickelt. Die vollständige Spezifikation von WML findet sich in [11].

WML eignet sich besonders für Endgeräte, die folgenden Beschränkungen unterliegen:

- Geräte mit kleiner Anzeige und/oder geringer Auflösung: Mobiltelefone bieten oft nur wenige Zeilen mit nur ca. zehn bis zwanzig Zeichen pro Zeile
- Geräte mit beschränkten Eingabemöglichkeiten: Mobile Endgeräte verfügen auf Grund der geringen Größe nur über eine kleine Tastatur, auf der gerade bei Mobiltelefonen die einzelnen Tasten mit mehreren Zeichen belegt sind. Oft ist auch keine Maus oder ein vergleichbares Zeigegerät vorhanden.
- geringe Rechenleistung im Vergleich zu Personal Computern: Die Ausstattung mit RAM und die Leistungsfähigkeit der CPU sind bei mobilen Endgeräten deutlich beschränkt.
- beschränkte Leistungsfähigkeit des Netzwerks: Das reibungslose Funktionieren der drahtlosen Datenübertragung kann unter geringer Bandbreite und großen Verzögerungen leiden. Übertragungsraten von 300 Bit/s bis 10 kBit/s und Verzögerungen von 5 bis 10 Sekunden sind nicht ungewöhnlich.

WML kann folgendermaßen umrissen werden:

- WML ermöglicht zum jetzigen Zeitpunkt die Übertragung von Informationen in Text- und Bildform. Es bietet mehrere Befehle, um das Layout und die Formatierung von Text und Bild festzulegen.

[1] XML ist in [10] spezifiziert. Die *Document Type Definition* (DTD) von WML 1.2 findet sich in [11, Kapitel 13.2] oder unter der URL `http://www.wapforum.org/DTD/wml_1.2.xml`.

- Die logische Einheit, mit der der Benutzer interagiert, ist die *WML-Card*. Eine Card ist vergleichbar mit einer HTML Seite und wird ebenfalls durch einen URI identifiziert. Diese Cards werden in *WML-Decks* gruppiert. Ein Deck ist die kleinste Einheit, die vom Server an den Client übertragen werden kann. Für genauere Informationen zu Decks und Cards siehe 2.1.

- WML erlaubt die Verwendung von Links, um dem Benutzer die Navigation zwischen den einzelnen Cards und die Ausführung von Scripts zu ermöglichen. Außerdem enthält es Befehle zur Behandlung von Ereignissen.

- Die WML-Cards können durch Variablen parametrisiert werden. Die Werte diese Variablen werden zur Laufzeit automatisch an der Stelle ihres Auftretens substituiert. Variablen behalten ihren Wert bei, wenn zu einer anderen Card gewechselt wird. Auf diese Weise kann der Inhalt einer Card vom Ergebnis einer Abfrage auf einer anderen Card abhängig gemacht werden, ohne dass die abgefragte Information an den Server übertragen werden muss.

Die gesamte WML-Information wird in kompakter binärkodierter Form über das drahtlose Netzwerk übermittelt.

2.1. Decks und Cards

WML gliedert Informationen in Einheiten mit den Bezeichnungen *Decks* und *Cards*. Der Benutzer navigiert durch eine Serie von Cards. Eine einzelne Card ist mit einer einzelnen HTML Seite im World Wide Web vergleichbar. Sie beinhaltet im Allgemeinen Textinformationen, Bilder, Eingabefelder und Links zu anderen Cards und kann durchaus größer als das Display des Endgerätes sein. Eine Card wird in WML durch das `card`-Element beschrieben.

Für das Deck gibt es im WWW keine direkte Entsprechung. Ein Deck ist eine Zusammenfassung von einer oder mehreren Cards zu einer Einheit, die als ein Paket vom Server an den Client übermittelt wird. Wenn der Benutzer von einer Card zu einer anderen wechselt, die dem gleichen Deck angehört, bleibt ihm die mitunter doch recht große Ladeverzögerung erspart, da sich die Card bereits im Speicher des Endgeräts befindet. Das Übertragen mehrerer Cards als Einheit ist vorteilhafter als das Übertragen jeder Card für sich, wenn man den Overhead der WAP-Protokolle

und die niedrige Bandbreite bedenkt. Ein Deck wird in WML durch das `wml`-Element beschrieben und muss die Definition mindestens einer Card beinhalten.

Bei diesem Konzept der Card als kleinste Informationseinheit, mit der der Benutzer interagiert, und des Decks als kleinste Übertragungseinheiten zwischen Server und Client kann es vorkommen, dass alle Cards innerhalb eines Decks gewisse Gemeinsamkeiten aufweisen sollen. Es kann zum Beispiel erwünscht sein, dass jede Card die Möglichkeit bieten soll, zur vorherigen Card zurückzukehren. Um diese Gemeinsamkeiten explizit machen zu können und um Speicherplatz zu sparen, kann ein Deck ein *Template* beinhalten, dessen Eigenschaften automatisch auf alle Cards übertragen werden. Jede Card hat allerdings die Möglichkeit, das durch das Template vorgegebene Standardverhalten neu zu definieren bzw. zu überschreiben. Ein Template wird in WML durch das `template`-Element beschrieben und muss in das `wml`-Element eingebettet sein, falls es benötigt wird.

Außerdem können für das ganze Deck noch Meta-Informationen angegeben und der Zugriff eingeschränkt werden. Diese Daten können in einem *Header* festgelegt werden, welcher, falls er benötigt wird, in das `wml`-Element eingebettet sein muss. Der Header wird in WML durch das `head`-Element beschrieben und muss mindestens ein `access`- oder `meta`-Element beinhalten, um gültig zu sein.

2.1.1. Grundlegender Aufbau eines WML-Dokumentes

Am Anfang eines WML-Dokuments muss immer eine XML-Deklaration und eine Dokumenttypendefinition stehen. Anschließend muss genau ein Deck mit dem `wml`-Element definiert werden. Dieses Element beinhaltet maximal ein `head`-Element, maximal ein `template`-Element und eine nicht leere Liste von `card`-Elementen, in genau dieser Reihenfolge. Ein Deck muss also mindestens eine Card definieren, um gültig zu sein; die Header- und Template-Informationen dürfen weggelassen werden, wenn sie nicht benötigt werden. Der allgemeine Aufbau eines WML-Dokumentes ist also folgender:

```
<?xml version="1.0"?>
<!DOCTYPE wml PUBLIC "-//WAPFORUM//DTD WML 1.2//EN"
                     "http://www.wapforum.org/DTD/wml_1.2.xml">
<!-- Das wml-Element enthält die Definition des Decks -->
<wml>
  <!-- Das head-Element ist optional -->
  <head>
```

```
    <!-- Angabe von Meta-Daten und/oder Zugriffsrestriktionen -->
  </head>

  <!-- Das template-Element ist optional -->
  <template>
    <!-- Grundeinstellungen für alle Cards des Decks -->
  </template>

  <!-- Die Definition mindestens einer Card ist notwendig! -->
  <card id="id_1" title="Titel Card 1">
    <!-- Inhalt und Verhalten der ersten Card -->
  </card>

  <!-- Die Definition weiterer Cards ist optional -->
  <card id="id_2" title="Titel Card 2">
    <!-- Inhalt und Verhalten der zweiten Card -->
  </card>
</wml>
```

Die bisher vorgestellten `head`-, `template`- und `card`-Elemente werden im Folgenden genauer erläutert.

2.1.2. Der Header

Wie zuvor schon kurz erwähnt, dient der Header dazu, den Zugriff auf das gesamte Deck zu beschränken und/oder Meta-Daten für das Deck anzugeben.

Der Zugriff kann mit Hilfe des `access`-Elementes in der Weise beschränkt werden, dass das Deck bzw. seine Cards nur dann angezeigt werden können, wenn sie von einer bestimmten URL (oder einer Menge von URLs) referenziert werden; das heißt nur dann, wenn der Benutzer von einer bestimmtem URL zum zugriffsgeschützten Deck navigiert.

Für die Angabe von Meta-Informationen dient das `meta`-Element, das entweder über ein `name`- oder ein `http-equiv`-Attribut verfügen muss.

Meta-Informationen, die das `name`-Attribut verwenden, müssen vom User Agent ignoriert werden. Wie bei HTML werden hier Informationen gespeichert, die zum Beispiel für Search Engines von Interesse sind. Sinnvolle Werte für das `name`-Attribut könnten also zum Beispiel „*keywords*“ oder „*generator*“ sein.

Die andere Variante, also Meta-Informationen, die das `http-equiv`-Attribut verwenden, dient dazu, HTTP-Header innerhalb von WML angeben zu können. Meta-

Daten dieser Art sollen in HTTP- oder WSP-Response-Header umgewandelt werden, bevor der Inhalt beim User Agent ankommt.

2.1.3. Das Template

Das `template`-Element deklariert ein Muster für alle Cards des Decks. Eine Angabe im Template ist identisch mit der gleichen Angabe in allen Cards; sie kann jedoch in den Cards wieder durch ein anderes Verhalten ersetzt werden. Das `template`-Element enthält unter anderem die drei Attribute `onenterforward`, `onenterbackward` und `ontimer`, mit denen das Verhalten beim Auftreten von bestimmten wesentlichen Ereignissen festgelegt werden kann. Alle drei Attribute benötigen als Wert einen URI, zu dem weitergeleitet wird, wenn das jeweilige Ereignis eintritt. Das `onenterforward`-Attribut gibt an, dass zum angegebenen URI weitergeleitet werden muss, sobald eine Card des Decks betreten wird. Mit `onenterbackward` wird festgelegt, dass zum angegebenen URI weitergeleitet werden muss, wenn zu einer Card des Decks zurückgekehrt wird. Das dritte `ontimer`-Attribut leitet zum angegebenen URI weiter, wenn eine bestimmte Zeit abgelaufen ist.

Zusätzlich oder alternativ zu diesen Attributen kann das `template`-Element eine Liste von `onevent`- und `do`-Elementen enthalten (siehe Kapitel 2.2 auf Seite 19). Mit dem `onevent`-Element kann man genau wie mit den oben angeführten Attributen auf das Eintreten eines Ereignisses reagieren. Die Reaktion kann dabei allerdings komplexer sein, als nur zu einem bestimmten URI weiterzuleiten; es kann beispielsweise auch der Wert einer Variablen verändert werden. Mit dem `do`-Element können Elemente zur Benutzerschnittstelle hinzugefügt werden, die der Benutzer aktivieren kann und die dann eine bestimmte Aktion auslösen. Beispielsweise kann ein Menü, das mit einer bestimmten Taste des mobilen Endgeräts geöffnet wird, um einen Menüpunkt erweitert werden.

2.1.4. Die Card

Eine Card enthält Text und Eingabeelemente zusammen mit der entsprechenden Layout-Information. Die Card ist die Einheit, mit der der Benutzer interagiert. Das heißt, der Benutzer navigiert von einer Card zur nächsten. Er kann auch in einer Card Informationen eingeben, die auf verschiedenste Weise weiterverarbeitet werden können. Die Informationen können durch Verwendung von Variablen, die ihren Wert beim Wechseln der Card beibehalten, in einer anderen Card weiterverwendet werden.

Die Informationen können aber auch an ein WMLScript (siehe 4 auf Seite 30) oder an einen Server übergeben werden.

WML lässt für die Implementierung, das heißt für die Anzeige und die Art der Eingabe von Informationen, große Freiheiten zu. Es werden also beispielsweise keine Anforderungen an Größe oder Auflösung der Anzeige oder an die Eingabemöglichkeiten gesetzt. So lassen sich etwa *Widgets* (grafische Symbole, die ein Interaktion zwischen dem Benutzer und dem Gerät erlauben) definieren, ohne dabei vorzuschreiben, wie auf dieses Widgets zugegriffen werden kann oder wie sie grafisch präsentiert werden sollen. Die Widgets können beispielsweise auch als Menüeinträge interpretiert werden.

Das `card`-Element enthält eine Reihe von Attributen, von denen manche schon vom Template (siehe 2.1.3 auf der vorherigen Seite) her bekannt sind. Das sind `onenterforward`, `onenterbackward` und `ontimer`.

Zusätzlich gibt es noch eine Reihe anderer Attribute, die für eine Card gesetzt werden können. Dem `id`-Attribut kommt bei der Card eine besondere Bedeutung zu: es kann als Anker (bzw. als Ziel) eines Links verwendet werden. Nur dadurch wird es möglich, eine bestimmte Card eines Decks in einer URL zu referenzieren. Der Identifier muss innerhalb des Decks eindeutig sein.

Das `title`-Attribut dient zur Angabe eines Titels für die Card. Der Titel kann, wie zuvor schon angedeutet, auf verschiedene Weise dargestellt werden. Er ist auch sinnvoll, wenn der Benutzer einen Bookmark auf die Card setzen will.

Das Attribut `newcontext` kann auf einen booleschen Wert (*true* oder *false*) gesetzt werden und gibt an, ob der Kontext des Browsers beim Betreten der Card zurückgesetzt werden soll. Wird dieses Attribut auf *true* gesetzt, so bedeutet das im Wesentlichen, dass beim Betreten der Card (durch einen `go`-Task) alle Variablen des Browsers und die Aufzeichnungen über zuvor besuchte Cards (die *navigational history*) gelöscht werden. Die Voreinstellung für dieses Attribut ist *false*.

Das `card`-Element kann verschiedene andere Elemente beinhalten, wobei eine bestimmte Reihenfolge eingehalten werden muss. Die Elemente sind der Reihe nach:

- eine möglicherweise leere Liste von `onevent`-Elementen: Mit diesem Element lassen sich Reaktionen auf Ereignisse (wie etwa `onenterbackward`) definieren, die bei deren Eintreten ausgeführt werden. Eine genauere Beschreibung der Events folgt in 2.2 auf der nächsten Seite.

- maximal ein `timer`-Element: Dieses Element dient zur Erzeugung eines Timers, der mit dem Betreten der Card gestartet wird. Siehe 2.3 auf Seite 22.
- eine möglicherweise leere Liste von `do`-, `p`- oder `pre`-Elementen: Mit dem `do`-Element lassen sich Aktionen definieren, die vom Benutzer manuell gestartet werden können (vgl. Kapitel 2.2). Das `p`-Element kennzeichnet einen Absatz und kann selbst Text, Tabellen, Eingabefelder und vieles mehr enthalten; das `pre`-Element dient zur Darstellung von vorformatiertem Text (für `p` und `pre` siehe 2.6 auf Seite 26).

2.2. Events und Tasks

Events sind Ereignisse, die durch bestimmte Aktionen des Benutzers oder durch bestimmte Definitionen innerhalb des Decks oder einer Card ausgelöst werden. Auf das Eintreten dieser Ereignisse kann durch die Definition von *Tasks* (Aufgaben) reagiert werden. Events können also an Tasks gebunden werden.

2.2.1. Übersicht über die Events

Es kann zwischen zwei Arten von Events unterschieden werden. Zunächst gibt es so genannte *intrinsic* Events, welche Zustandsänderungen im User Agent anzeigen. Diese Events haben definierte Namen und werden nun kurz vorgestellt:

`ontimer`: Dieser Event wird ausgelöst, wenn ein Timer abläuft. Diese Timer können mit dem `timer`-Element in den Cards definiert werden und beginnen zu laufen, wenn die betreffende Card betreten wird.

`onenterforward`: Dieser Event wird ausgelöst, wenn eine Card über den `go`-Task (oder eine Methode mit identischer Semantik) betreten wird.

`onenterbackward`: Die Rückkehr zu einer bereits zuvor besuchten Card (durch den `prev`-Task) löst dieses Ereignis aus.

`onpick`: Dieser Event wird ausgelöst, wenn der Benutzer ein Element einer Auswahlliste auswählt.

Die ersten drei Events dieser Gruppe, also `ontimer`, `onenterforward` und `onenterbackward`, können an einen Task gebunden werden, indem sie entweder als gleichnamiges Attribut des `card`- oder `template`-Elementes angegeben werden oder indem

sie im `type`-Attribut des `onevent`-Elementes eingetragen werden. Der vierte Event, also `onpick`, wird über das gleichnamige Attribut des `option`-Elementes gebunden.

Die zweite Art von Events wird vom Benutzer bewusst ausgelöst. Dazu können Widgets (grafische Elemente zur Interaktion) in die Benutzerschnittstelle eingebunden werden, die vom Benutzer zu einem beliebigen Zeitpunkt aktiviert werden können und dann den betreffenden Event zur Folge haben. Diese Widgets werden mit dem `do`-Element definiert und gleichzeitig an einen Task gebunden.

2.2.2. Übersicht über die Tasks

Es gibt vier Tasks, die an Events gebunden werden können. Diese Tasks werden in WML durch die gleichnamigen Elemente deklariert:

`go`: Dieser Task dient zur Weiterleitung an eine URL oder einen URI.

`prev`: Mit diesem Task kann zur zuletzt besuchten Card zurückgekehrt werden.

`noop`: Dieser Task steht für *no operation* und macht gar nichts. Er eignet sich dazu, `do`- oder `onevent`-Elemente (bzw. die entsprechenden Attribute), die im Template angegeben wurden, aus einer Card zu löschen.

`refresh`: Dieser Task kennzeichnet die Aktualisierung von Variablen und hat den Neuaufbau der Anzeige zur Folge.

Das `go`-Element darf eine Liste von `postfield`- oder `setvar`-Elementen (siehe Seite 23) beinhalten. Mit `postfield` wird ein Feldname und ein Wert angegeben, der an den Server übertragen werden soll. Mit dem `setvar`-Element können zusätzlich zur Weiterleitung an den URI Variablen des WML-Browsers verändert werden. Die wichtigsten Attribute des `go`-Elementes sind:

`href`: Dieses Attribut gibt den Ziel-URI an.

`sendreferer`: kann auf einen booleschen Wert gesetzt werden und gibt an, ob dem Server im HTTP „Referer“-Header der URI des Decks übermittelt werden soll, von dem aus der Benutzer zum Ziel-URI navigiert; die Voreinstellung ist hier *false*.

`method`: Mögliche Werte sind `post` und `get`. Hier kann also angegeben werden, ob ein HTTP post- oder get-Request an den Server übermittelt werden soll. Die Voreinstellung ist `get`.

Das folgende Beispiel zeigt, wie das `postfield`-Element innerhalb des `go`-Elementes verwendet werden kann, um Daten an den Server zu übertragen:

```
<go method="post" href="http://domain/path">
  <postfield name="name_1" value="wert_1">
  <postfield name="name_2" value="wert_2">
</go>
```

2.2.3. Bindung von intrinsic Events in Attributen

Wie schon in den Unterabschnitten 2.1.3 und 2.1.4 auf Seite 17 ausgeführt, besitzen das `template`- und das `card`-Element die Attribute `ontimer`, `onenterforward` und `onenterbackward`. Diese Attribute können verwendet werden, um die gleichnamigen intrinsic Events zu binden. Der Wert der Attribute ist dabei jeweils ein URI, an den beim Eintreten des Ereignisses weitergeleitet werden soll. Dies entspricht quasi der Bindung an einen leeren `go`-Task, bei dem nur das `href`-Attribut angegeben ist.

Das Gleiche gilt für das `onpick`-Attribut des `option`-Elementes; auch hier muss der Wert ein URI sein.

2.2.4. Bindung von intrinsic Events durch das `onevent`-Element

Die intrinsic Events können auch mit dem `onevent`-Element an einen Task gebunden werden. Mit dieser Schreibweise ergibt sich eine größere Freiheit in der Auswahl der durchzuführenden Aufgabe. Das `onevent`-Element hat ein `type`-Attribut, welches auf den Namen eines intrinsic Events gesetzt werden muss (also zum Beispiel `<onevent type="ontimer">`). Es muss außerdem genau ein Element beinhalten, das einem der vier Tasks zugeordnet ist.

Das `onevent`-Element darf innerhalb der Elemente `template`, `card` und `option` verwendet werden.

2.2.5. Erstellung von Widgets mit dem `do`-Element

Das `do`-Element bietet die Möglichkeit, so genannte Widgets in das Benutzerinterface zu integrieren. Über diese Widgets wird in der Spezifikation nicht viel mehr

ausgesagt, als dass sie in irgendeiner Weise vom Benutzer aktiviert werden können. Die Aktivierung entspricht einem Event, der ebenfalls durch das `do`-Element an einen bestimmten Task gebunden wird.

Das `do`-Element selbst muss eines der vier Task-Elemente beinhalten, das bei Aktivierung des Widgets ausgeführt wird. Die wichtigsten Attribute dieses Elementes sind:

`type`: Dieses Attribut gibt dem User Agent einen Hinweis darauf, wie das Widget benutzt werden kann und welche grafische Präsentationsform daher gewählt werden kann. Mögliche Werte sind z. B. `accept` für eine positive Bestätigung, `prev` für die Rückkehr zur zuletzt besuchten Card oder `help` für die Anforderung von Hilfe.

`label`: Der User Agent sollte diesen Wert als Text des Widgets anzeigen. Der Label sollte nicht länger als sechs Zeichen sein.

`name`: `do`-Elemente können benannt werden. Wird dieses Attribut nicht angegeben, so wird der Wert von `type` verwendet. Ein `do`-Element einer Card ersetzt ein `do`-Element des Templates gleichen Namens.

2.3. Timers

Mit dem `timer`-Element lässt sich ein Timer auf Card-Ebene definieren, der zu laufen beginnt, sobald die Card betreten wird. Der Timer ist als Zähler implementiert, der ab einem positiven Wert rückwärts zählt, bis der Wert Null erreicht wird. Der Übergang vom Zählerstand Eins auf Null löst dann den `ontimer`-Event aus, der im Abschnitt 2.2 auf Seite 19 beschrieben wurde. Ein laufender Timer wird durch die Ausführung eines beliebigen Tasks angehalten.

Die Timer haben eine zeitliche Auflösung von etwa einer Zehntelsekunde. Diese Angabe darf nicht zu genau genommen werden, da lediglich die *idle time* des User Agents gemessen wird, also die Zeit, in der er keine Aufgaben zu erledigen hat.

2.4. Variablen und Eingabefelder

WML unterstützt Variablen, deren Wert vom User Agent zur Laufzeit an der Stelle ihrer Referenzierung substituiert wird. Eine Referenzierung darf im laufenden Text

einer Card und als Wert gewisser Element-Attribute auftreten. Dadurch wird es möglich, die Anzeige einer Card dynamisch, auf der Grundlage von Benutzereingaben, zu ändern, ohne dass der Web-Server die Card neu generieren muss. Variablen dürfen insbesondere auch in `href`-Attributen (im `go`-Task, etc.) referenziert werden, was eine dynamische Navigation zwischen den Cards ermöglicht.

Variablen haben einen Namen und einen Wert in Form einer einfachen Zeichenkette. Der Wert kann durch `$`*`varname`* oder `$(`*`varname`*`)` referenziert werden.

2.4.1. Die Lebensdauer von Variablen

Der User Agent kann einen Mechanismus zur Verwaltung von Bookmarks oder zur Eingabe von URLs anbieten. Wenn auf diese Weise eine neue Card besucht wird, hat der User Agent zuvor alle Variablen zu löschen. Dies gilt generell, wenn die Navigation zu einer neuen Card nicht aus dem Kontext der zuletzt besuchten Card resultiert. Dadurch wird verhindert, dass Variablen in einer Card sichtbar sind, die im Kontext dieser Card keinen oder möglicherweise sogar einen falschen Sinn ergeben. Auch sensitive Informationen können so nicht versehentlich in falsche Hände gelangen.

Darüber hinaus besitzt das `card`-Element das `newcontext`-Attribut, wie bereits in 2.1.4 auf Seite 17 dargelegt wurde. Wird dieses Attribut auf den Wert *true* gesetzt, muss der Browser vor dem Betreten der Card ebenfalls alle Variablen löschen.

2.4.2. Zuweisung von Werten an Variablen

Variablen können auf verschiedene Arten gesetzt werden:

- durch das `setvar`-Element als Seiteneffekt eines `go`-, `prev`- oder `refresh`-Tasks: Das `setvar`-Element besitzt die Attribute `name` und `value`, mit denen der Name der zu ändernden Variablen und ihr neuer Wert angegeben werden können. Als Wert kann auch eine andere Variable referenziert werden.
- durch die Verwendung von Eingabefeldern, die im Folgenden beschrieben werden
- Variablen können außerdem durch ein WMLScript initialisiert oder verändert werden.

Für Benutzereingaben stehen zwei Elemente zur Verfügung, das `input`- und das `select`-Element:

1. Das `input`-Element weist die Benutzereingabe einer Variablen zu, die durch das `name`-Attribut benannt wird. Mit dem `value`-Attribut lässt sich eine Voreinstellung für den Wert angeben, den der Benutzer bestätigen oder ändern kann. Mit weiteren Attributen können der Typ (Text oder Passwort), das Format (Ziffern, Großbuchstaben, etc.), ein Titel und andere Zusatzinformationen angegeben werden.

2. Das `select`-Element erlaubt dem Benutzer die Auswahl einer oder mehrerer Optionen aus einer Liste vordefinierter Möglichkeiten. Dabei kann gewünscht sein, dass der Benutzer genau eine der angebotenen Optionen auswählen muss oder dass die gleichzeitige Auswahl mehrerer Optionen erlaubt ist. Auch hier wird das `name`-Attribut verwendet, um die Variable zu benennen, in der das Ergebnis abgelegt werden soll. Bei Mehrfachauswahl ist der Variablenwert eine durch Semikolon getrennte Liste der ausgewählten Optionen.

   Das `select`-Element muss eine Liste von `option`- oder `optgroup`-Elementen beinhalten, die nicht leer sein darf. Jedes `option`-Element beschreibt eine einzelne Auswahlmöglichkeit. Das `optgroup`-Element muss selbst wieder eine nicht leere Liste von `option`- oder `optgroup`-Elementen beinhalten und kann zur Gruppierung verwandter Elemente oder zur Erstellung komplexer Hierarchien verwendet werden.

2.4.3. Gruppierung von Eingabefeldern

Mit dem `fieldset`-Element lassen sich `input`- und `select`-Eingabefelder und Text, die in semantischer Verbindung zueinander stehen, zu logischen Gruppen zusammenfassen. Diese Informationen können vom User Agent verwendet werden, um das Layout und die Navigation zu optimieren.

Das `fieldset`-Element darf neben Text und Texthervorhebungen (`em`, `strong`, `b`, `i`, `u`, `big` und `small`) noch folgende Elemente beinhalten: Eingabefelder (`input`, `select` und verschachtelte `fieldset`-Elemente), Links (`a` und `anchor`), Bilder (`img`), Zeilenwechsel (`br`), Tabellen (`table`) und Widget-Definitionen (`do`).

2.5. Links

In WML können Links mit den Elementen `anchor` und `a` erstellt werden. Das `anchor`-Element ist das allgemeinere der beiden; das vermutlich gebräuchlichere und von HTML her vertrautere `a`-Element kann als Kurzschreibweise des `anchor`-Elementes betrachtet werden.

Links dürfen (in beiden Schreibweisen) fast überall eingefügt werden, wo formatierter Text erlaubt ist, also in Absätzen, Zellen von Tabellen, innerhalb von Texthervorhebungen, und so weiter.

Mit dem `anchor`-Element wird die Quelle eines Links spezifiziert. Das `anchor`-Element muss einen Task beinhalten, der angibt, was passieren soll, wenn der Link vom Benutzer verfolgt wird. Dafür muss einer der Tasks `go`, `prev` oder `refresh` verwendet werden. Neben diesem Task darf das Element auch noch Text und die Elemente `br` und `img` beinhalten. Außerdem darf das `anchor`-Element noch ein `title`-Attribut besitzen, welches den Link identifizieren soll. Die Länge dieses Attributes sollte sechs Zeichen nicht überschreiten, damit es von allen User Agents korrekt dargestellt werden kann.

An Stelle der Kombination von `anchor`-Element und leerem `go`-Task kann das `a`-Element als Kurzschreibweise verwendet werden. Neben dem `title`-Attribut besitzt das `a`-Element noch das `href`-Attribut, das wie beim `go`-Task das Ziel des Links angibt.

2.5.1. URLs und fragment anchors

WML verwendet *Uniform Resource Locators* (URLs) in der gleichen Weise wie das World Wide Web zur Benennung und Referenzierung von Inhalten bzw. Dokumenten. Sie werden in WML sowohl für die Navigation als auch für die Referenzierung externer Resourcen verwendet. Externe Resourcen können Bilder oder auch WMLScript-Kompilierungseinheiten sein. URLs sind in RFC2396 ([12]) definiert.

In WML kommt dem *fragment anchor*[2] einer URL eine besondere Bedeutung zu. Das URL-Fragment wird hier zur Referenzierung einer Card innerhalb eines Decks verwendet. Wenn kein Fragment angegeben ist, bezieht sich die URL auf das gesamte Deck bzw. dessen erste Card. Innerhalb eines Decks können relative URLs verwendet

[2] In HTML können mittels `<a name="...">` Zielanker für Hyperlinks innerhalb des Dokumentes erstellt werden. Diese Anker können dann im Fragment einer URL (nach dem `#`-Zeichen) referenziert werden, um genau an die betreffende Stelle im Dokument gelangen zu können.

werden, um andere Cards im gleichen Deck zu benennen — dabei wird bei der URL also *nur* das Fragment angegeben.

URLs können auch verwendet werden, um WMLScript-Funktionen aufzurufen. Dabei benennt die URL die WMLScript-Kompilierungseinheit, die die aufzurufende Funktion beinhaltet. Der Funktionsname und die Parameterliste werden im Fragment der URL, also nach dem `#`-Zeichen, angegeben, wobei die Parameter in runde Klammern eingeschlossen werden müssen.

2.6. Textformatierung

In WML existieren verschiedene Mechanismen zur Textformatierung. Durch sie lassen sich Absätze und Tabellen, manuelle Zeilenwechsel und verschiedene Texthervorhebungen (Fettdruck, etc.) definieren. Die entsprechenden Elemente werden im Folgenden beschrieben:

Absätze: Eine Card darf Absätze beinhalten, die wiederum Text und andere Elemente enthalten können. Ein Absatz wird mit dem `p`-Element gekennzeichnet. Mit dem `align`-Attribut wird angegeben, ob der Absatz linksbündig, rechtsbündig oder zentriert gesetzt werden soll. Aufeinanderfolgende *white space characters* (das sind Leerräume wie Leerzeichen, Tabulatoren oder Zeilenwechsel) in einem Absatz werden zu einem einzelnen Leerzeichen zwischen den Worten zusammengefasst.

Manuelle Zeilenwechsel: Ist ein Zeilenwechsel an einer bestimmten Stelle im Text (oder nach einem Element) beabsichtigt, kann das `br`-Element verwendet werden. Der User Agent muss an dieser Stelle eine neue Zeile beginnen (bzw. sollte es innerhalb von Tabellen wenigstens versuchen).

Tabellen: WML unterstützt einfache Tabellen. Einfach bedeutet, dass eine Tabelle keine anderen Tabellen beinhalten darf. Auch können keine benachbarten Zellen zu einer einzigen verbunden werden, wie es in HTML möglich ist. Spaltenbreiten, Zeilenhöhen, Abstände zwischen einzelnen Zeilen oder Spalten, Spaltenüberschriften und Rahmenattribute können ebenfalls nicht gesetzt werden. Es lässt sich also tatsächlich nur eine tabellarische Struktur definieren, deren Zellen Text und Bilder beinhalten dürfen.

Eine Tabelle wird durch das `table`-Element definiert. Zur Definition der Zeilen und Zellen der Tabelle werden wie in HTML die Elemente `tr` und `td` verwendet.

Texthervorhebungen: Zur Hervorhebung von Textpassagen können die Elemente `em` zur Hervorhebung (*emphasis*), `strong` für starke Hervorhebung (*strong emphasis*), `i` für Kursivschrift (*italic font*), `b` für Fettdruck (*bold font*), `u` zum Unterstreichen (*underline*), `big` für eine größere Schriftart (*large font*) und `small` für eine kleinere Schriftart (*small font*) verwendet werden. Man beachte, dass `em` und `strong` abstrakte Angaben sind, während alle anderen Texthervorhebungen konkreter Natur sind. Die abstrakten Angaben sollten bevorzugt verwendet werden.

2.7. Bilder

Innerhalb von Absätzen (`p`), Zellen von Tabellen (`td`), Links (`a` und `anchor`), Feldgruppierungen (`fieldset`) und Texthervorhebungen (`em`, `strong`, `b`, `i`, `u`, `big` und `small`) dürfen Bilder eingefügt werden. Dazu dient das `img`-Element, dessen zwingend erforderliche Attribute folgende sind:

`alt`: Dieses Attribut dient zur Angabe einer alternativen Textrepräsentation für den Fall, dass das Bild nicht geladen oder vom User Agent nicht dargestellt werden kann.

`src`: Hier wird der URI des Bildes angegeben. Wie bei einem Link kann auch hier eine Variable referenziert werden.

Das *Wireless Application Environment* (WAE) erlaubt prinzipiell die Verwendung von Standardbildformaten, wie zum Beispiel PNG (*Portable Network Graphics*) oder GIF (*Graphics Interchange Format*). Zusätzlich spezifiziert das WAE aber ein eigenes, für den drahtlosen Bereich optimiertes Bildformat, das *Wireless Bitmap* Format, welches im folgenden Kapitel beschrieben wird. Jeder WML-Browser, der Grafiken anzeigen kann, muss zumindest den Typ 0 des *Wireless Bitmap* Formats unterstützen. WBMP ist daher das Bildformat, das man in WML-Decks am häufigsten antreffen wird.

Der User Agent muss dem WAP-Server bei der Initialisierung der Sitzung mitteilen, welche Bildformate er unterstützt. Er tut dies, indem er den Standard HTTP- bzw. WSP-Header `Accept` setzt, wie das folgende Beispiel zeigt:

```
Accept: image/vnd.wap.wbmp; level=0
```

Die Angabe `level=0` bedeutet, dass der Typ 0 von WBMP akzeptiert wird.

3. Das Wireless Bitmap Format

Das Wireless Bitmap (WBMP) Format ist in [13, Kapitel 6 und Anhang A] spezifiziert. Es handelt sich bei WBMP genau genommen um eine Kapselung von mehreren Bildformaten bzw. Typen. Das Format ist erweiterbar und an die Bedürfnisse der Clients im drahtlosen Bereich angepasst.

Das WBMP-Format besteht aus einem generischen Header, der Informationen beinhaltet, die alle Bildformate besitzen, und aus einer typabhängigen Formatspezifikation. Der Header beinhaltet Typ-Nummer, Breite und Höhe des Bildes und die WBMP-Versionsnummer. Die restliche Bildinformation (inklusive eventuell notwendiger Farbpaletten oder anderer Zusatzinformationen) steckt im typabhängigen Teil des Formates.

In der Version 1.2 des Dokumentes [13] ist lediglich der Typ 0 des Wireless Bitmap Formats definiert. Der WBMP Typ 0 hat folgende Eigenschaften:

- keine Komprimierung der Bilddaten
- ein Bit Farbtiefe pro Bildpunkt: Weiß = 1 und Schwarz = 0
- Acht Bildpunkte werden zu einem Byte zusammengefasst, wobei der äußerst linke Bildpunkt dem höchstwertigigen Bit entspricht.
- Das Bild wird zeilenweise von oben nach unten abgespeichert. Einzelne Zeilen werden von links nach rechts gespeichert. Ist die Bildbreite nicht durch acht teilbar, werden an jedem Zeilenende Null-Bits angefügt, sodass eine Zeile immer durch ganze Bytes dargestellt wird.

4. WMLScript

WMLScript ist eine einfache Scriptsprache zur Ergänzung von WML. Ein WML-Script kann von einem WML-Deck aufgerufen werden und bietet unter anderem folgende Möglichkeiten:

- Überprüfung der Gültigkeit von Benutzereingaben am Client
- einfache Berechnungen, Konvertierung von Währungen, Maßeinheiten und dergleichen
- schnellere Anzeige von Fehlermeldungen, Bestätigungen und Warnungen, ohne mit dem Server interagieren zu müssen
- WMLScript kann auch benutzt werden, um den User Agent zu steuern. Ist der User Agent zum Beispiel ein Handy, so erhält man eventuell Zugriff auf das Adressverzeichnis oder kann das Handy dazu veranlassen einen Anruf zu tätigen.[1]

Ein WMLScript-Quellcode muss zuerst in Bytecode übersetzt werden, bevor das Script an den Client geschickt werden kann. Diese Compilierung übernimmt üblicherweise das Wap-Gateway.

4.1. Die lexikalische Struktur

Die erste Stufe bei der Compilierung einer Sprache ist die lexikalische Analyse (siehe [14]). Sie beschäftigt sich noch nicht mit der Syntax der Sprache, sondern zerlegt den Text in so genannte *Tokens*, welche die Elemente der Sprache abstrahieren. Die lexikalische Analyse entfernt *white space* und Sourcecode-Kommentare und erkennt Literale, Bezeichner und reservierte Wörter.

[1] Diese Funktionen sind allerdings nicht grundlegender Bestandteil von WMLScript selbst, sondern werden in der *Wireless Telephony Application* spezifiziert. Siehe Kapitel 5 auf Seite 45.

Im Folgenden wird beschrieben, wie die genannten Elemente in WMLScript auszusehen haben.

WMLScript verfügt über die gleiche Art von Sourcecode-Kommentaren wie die Programmiersprache C++, Block- und Zeilenkommentare:

- Der Block-Kommentar beginnt mit der Zeichenfolge `/*` und endet mit `*/`. Block-Kommentare dürfen nicht verschachtelt werden.
- Der Zeilen-Kommentar wird durch `//` eingeleitet und reicht bis zum Zeilenende. Zeilenkommentare dürfen innerhalb eines Block-Kommentars auftreten.

Literale, das sind Textrepräsentationen von Werten der Datentypen einer Sprache, werden in WMLScript, wie folgt, angegeben:

- Ganzzahlige Literale können dezimal, hexadezimal oder oktal angegeben werden. Wie in der Programmiersprache C wird der hexadezimalen Darstellung die Zeichenfolge `0x` vorangestellt und die Ziffern werden um die Buchstaben `A` bis `F` mit den Werten von 10 bis 15 erweitert. Die oktale Darstellung beginnt mit der Ziffer `0` und darf nur die Ziffer `0` bis `7` beinhalten. Beispiele für die drei Darstellungsarten sind `4711`, `0x1267` und `011147`.
- Fließkommazahlen dürfen einen Dezimalpunkt und/oder einen Exponenten beinhalten. Der Exponent (zur Basis 10) wird mit dem Zeichen `e` oder `E` eingeleitet. Beispiele sind `3.1415`, `31.415e-1` und `31415e-4`.
- String-Literale werden durch die Zeichen `"` oder `'` eingeschlossen. Zwischen den beiden Anführungszeichen müssen manche Zeichen mit einem Backslash escaped werden, wie man das von der Programmiersprache C her kennt. Dies sind `\'`, `\"`, `\\`, `\/`, `\b`, `\f`, `\n`, `\r` und `\t`. Beispiele für String-Literale sind `"String-Literal"`, `'Farbe: \'rot\''` oder `"Newline = \\n"`.
- Für boolesche Werte gibt es die beiden Literale `true` und `false`.
- Außerdem gibt es in WMLScript noch das spezielle Literal `invalid`, das einen ungültigen Wert darstellt.

Bezeichner (*Identifier*) für Variablen- und Funktionsnamen und Pragmata (Siehe 4.2) können in WMLScript aus Groß- und Kleinbuchstaben, Ziffern und dem Underscore-Zeichen (`_`) zusammengesetzt werden; sie dürfen allerdings nicht mit einer Ziffer beginnen.

4.2. Pragmata

Pragmata sind Informationen zur jeweiligen Quelldatei bzw. Compilierungseinheit. Sie müssen vor den Funktionsdefinitionen stehen und werden mit dem Schlüsselwort „`use`“ eingeleitet. Es gibt drei verschiedene Arten von Pragmata für *externe Compilierungseinheiten*, *Zugriffssteuerung* und *Meta-Informationen*.

Diese Pragmata haben die folgende Syntax und Bedeutung:

- Externe Compilierungseinheiten bzw. *external compilation units*: Dieses Pragma hat die Form `use url` *`Identifier String-Literal`*`;` und wird verwendet, um einer externen Compilierungseinheit einen Namen zu geben. Mit diesem Namen können dann externe Funktionen dieser Quelldatei aufgerufen werden. Ein Beispiel für dieses Pragma ist im nächsten Abschnitt 4.3 über Funktionen zu finden.

- Zugriffssteuerung bzw. *access control*: Dieses Pragma wird mit `use access` eingeleitet und mit `domain` *`String-Literal`*, `path` *`String-Literal`* oder beidem fortgesetzt. Das Pragma legt fest, welche URLs auf die externen Funktionen dieser Quelldatei zugreifen dürfen, wobei mit `domain` ein Suffix einer Domäne und mit `path` ein Präfix eines Pfades angegeben werden. Die folgenden Beispiele sollen dies verdeutlichen:

```
// Zugriff nur von foo.com, www.foo.com u. dgl.
// Der Pfad ist innerhalb der Domäne ist egal.
use access domain "foo.com";

// Zugriff nur von /cgi-bin, /cgi-bin/test u. dgl.
// Egal von welcher Domäne.
use access path "/cgi-bin";

// Zugriff nur von foo.com, www.foo.com u. dgl.
// Und(!) nur von /cgi-bin, /cgi-bin/test u. dgl.
use access domain "foo.com" path "/cgi-bin";
```

- Meta-Informationen bzw. *meta information*: Es gibt drei verschiedene Meta-Pragmata mit folgenden Formen:

  Name: gibt Meta-Informationen für den Web Server an, z. B.: `use meta name "Created" "02/02/2002";`

HTTP equiv: Informationen dieser Art sollen in einen *WSP* oder *HTTP Response Header* konvertiert werden, z. B.: `use meta http equiv "Cache-Control" "private";`

User agent: Diese Informationen werden an den User Agent übermittelt. Beispiel: `use meta user agent "Type" "Test";`

4.3. Funktionen und Bibliotheken

Eine WMLScript-Quelldatei besteht aus einer oder mehreren Funktionsdefinitionen. Diese Funktionen sind Einheiten, die aus anderen Funktionen oder aus einem WML-Deck aufgerufen werden können. Eine Funktion kann parametrisiert sein und gibt einen Wert als Resultat zurück. Darüber hinaus kann sie Seiteneffekte haben, sie kann beispielsweise den Wert einer WML-Variablen ändern oder mit dem Benutzer interagieren. WMLScript ist daher eine prozedurale Programmiersprache.

Eine Funktionsdefinition wird mit `function` bzw. `extern function` eingeleitet. Dann folgen ihr Name, die Liste der Parameter und ein Block von Anweisungen. Das folgende Beispiel zeigt eine einfache, gültige Funktion:

```
extern function product(a, b) {
  return a * b;
}
```

Funktionen müssen folgenden Kriterien gerecht werden:

- Funktionsdefinitionen dürfen nicht verschachtelt werden.
- Der Name einer Funktion muss innerhalb der Quellcodedatei eindeutig sein.
- Beim Aufruf einer Funktion muss die Anzahl der übergebenen Argumente mit der Anzahl der formalen Parameter der Funktion übereinstimmen.
- Eine Funktion hat immer einen Rückgabewert. Wenn im Programmfluss keine explizite Rückgabe eines Wertes erfolgt, wird implizit ein leerer String, also `""` zurückgegeben.
- Die Parameter werden immer als Werte an die Funktion übergeben, nicht als Namen, Referenzen oder dergleichen (*Call-by-value*).

Unter einer Bibliothek versteht man in WMLScript eine benannte Sammlung von Funktionen. Bibliotheken können vom Programmierer nicht erstellt, sondern nur verwendet werden. Die Bibliotheken müssen am User Agent zur Verfügung stehen. WMLScript schreibt einen Satz von Standardbibliotheken vor, die im Unterabschnitt 4.9 beschrieben sind. Darüber hinaus können am User Agent noch proprietäre Bibliotheken des Herstellers zur Verfügung stehen; der Programmierer muss bei ihrer Verwendung aber sicherstellen, dass der Code nur auf einem kompatiblen Gerät ausgeführt wird.

Funktionen können auf verschiedene Weise aufgerufen werden, je nachdem, ob sie *lokale Funktionen* (in der gleichen Quelldatei definiert), *externe Funktionen* (in einer anderen Quelldatei definiert) oder *Bibliotheksfunktionen* sind.

- Lokale Funktionen werden mit ihrem Namen und den Argumenten aufgerufen. Beispiel: `var result = product(x, 23);`
- Bei externen Funktionen muss zusätzlich der Name der externen Quelldatei angegeben werden. Dazu wird das `use`-Pragma verwendet, auf das später noch näher eingegangen wird. Beispiel:

```
use url ExtScript "http://www.foo.com/product.wmls";

function test(p) {
  return ExtScript#product(17, p);
}
```

- Einer Bibliotheksfunktion wird der Name der Bibliothek gefolgt von einem Punkt vorangestellt. Das folgende Beispiel ruft eine Funktion namens `sqrt` der Bibliothek `Float` auf: `var squareroot = Float.sqrt(x);`

4.4. Anweisungen

Eine Funktion beinhaltet einen Block von Anweisungen (*Statements*). Diese Anweisungen werden beim Aufruf der Funktion in der Reihenfolge ihres Auftretens ausgeführt. Es gibt folgende Anweisungen[2]:

Block: Ein Block besteht aus einer Liste von Anweisungen innerhalb von geschwungenen Klammern.

[2]Die Anweisungen werden hier nicht sehr ausführlich erklärt, da sie von den verschiedensten Programmiersprachen her in ähnlicher Form vertraut sind.

Variablenanweisung: das Schlüsselwort `var`, gefolgt von einer durch Komma getrennten Liste von Variablendeklaration mit optionaler Initialisierung, zum Beispiel: `var i=17, j, k="ka";`

Leere Anweisung: Die leere Anweisung besteht lediglich aus einem Strichpunkt.

Ausdruck-Anweisung: Ein Ausdruck (siehe nächster Abschnitt 4.5) kann als Anweisung stehen. Beispiel: `x += 23;`

If-Anweisung: Sie hat, wie die Programmiersprache C, die beiden Formen `if (`*`Ausdruck`*`) `*`Anweisung`* oder `if (`*`Ausdruck`*`) `*`Anweisung`* `else` *`Anweisung`*. Bei der zweiteren Form ist das `else` immer an das letzte freie `if` gebunden. Wenn die Auswertung des Ausdrucks `true` ergibt, wird die erste Anweisung ausgeführt, ansonsten die zweite (nur sofern der `else`-Teil angegeben wurde).

While-Anweisung: Die While-Anweisung hat die Form `while (`*`Ausdruck`*`) `*`Anweisung`*. Die Anweisung wird also so lange ausgeführt, wie der Ausdruck `true` ergibt.

For-Anweisung: Die For-Anweisung hat die Form `for (`*`Ausdruck`*`; `*`Ausdruck`*`; `*`Ausdruck`*`) `*`Anweisung`*, wobei alle drei Ausdrücke optional sind. Statt des ersten Ausdrucks darf auch das Schlüsselwort `var` gefolgt von einer Variablendeklarationsliste stehen. Wie in C wird zunächst der erste Ausdruck evaluiert. Dann wird die Anweisung so lange ausgeführt, wie der zweite Ausdruck `true` ergibt. Nach jeder Ausführung der Anweisung wird außerdem der dritte Ausdruck ausgewertet.

Continue-Anweisung: Das Schlüsselwort `continue` kann innerhalb eines Blockes einer For- oder While-Anweisung verwendet werden und bewirkt, dass die Ausführung des Blockes an der Stelle der Continue-Anweisung abgebrochen und mit der nächsten Iteration der Schleife fortgefahren wird. Im Falle der For-Anweisung wird vor der nächsten Iteration noch der dritte Ausdruck ausgewertet.

Break-Anweisung: Wie die Continue-Anweisung darf das Schlüsselwort `break` nur innerhalb einer For- oder While-Anweisung verwendet werden. Sie unterbricht die Schleife und setzt die Programmausführung mit der nächsten Anweisung nach der Schleife fort.

Return-Anweisung: Die Return-Anweisung hat die Form `return` *`Ausdruck`*`;` oder einfach nur `return;` und beendet die Ausführung einer Funktion. Wenn ein Ausdruck angegeben ist, wird dieser ausgewertet und das Ergebnis an die aufrufende Funktion zurückgegeben. Anderenfalls wird ein leerer String (das String-Literal `""`) zurückgegeben. Das Gleiche gilt, wenn im Programmfluss das Ende des Funktionsblockes erreicht wird, ohne dass eine Return-Anweisung gefunden wurde.

4.5. Ausdrücke und Operatoren

Ausdrücke werden zur Laufzeit ausgewertet bzw. berechnet und haben einen Wert als Ergebnis. Die Werte können in anderen Ausdrücken oder Anweisungen verwendet werden oder als Argumente an Funktionen übergeben werden. Wie im letzten Unterabschnitt beschrieben wurde, benötigen manche Anweisungen Ausdrücke, zum Beispiel die Schleifenbedingungen der For- und While-Anweisungen.

Es gibt *einfache* und *komplexe* Ausdrücke. Einfache Ausdrücke sind Konstanten (bzw. Literale) und Variablen (auch Funktionsparametern). Die Auswertung eines einfachen Ausdrucks zur Laufzeit liefert einfach den Wert der Konstanten bzw. den aktuellen Wert der Variablen. Komplexe Ausdrücke sind Funktionsaufrufe (siehe 4.3) und Ausdrücke, die mit Hilfe von Operatoren aus anderen Ausdrücken zusammengesetzt werden. Eine Funktion wird zur Laufzeit des Programms ausgeführt und der Wert des Ausdruckes ist dann der Rückgabewert der Funktion. Bei Operatoren wird der Wert des Ausdrucks durch die jeweilige Operation und die Auswertung der Einzelausdrücke (der *Operanden*) berechnet.

In WMLScript stehen folgende Operatoren zur Verfügung:

Zuweisungsoperatoren: Ausdrücke mit diesen Operatoren haben die Form *`Bezeichner Zuweisungsoperator Ausdruck`*. Der Bezeichner muss eine Variable benennen, an die das Ergebnis der Operation zugewiesen wird. Das Ergebnis des Gesamtausdruckes ist mit dem Ergebnis der Operation identisch. An Zuweisungsoperatoren stehen `=`, `+=`, `-=`, `*=`, `/=`, `div=`, `%=`, `<<=`, `>>=`, `>>>=`, `&=`, `^=` und `|=` zur Verfügung. Die *`op`*`=`-Operatoren sind lediglich Abkürzungen; an Stelle von *`Bezeichner op= Ausdruck`* kann auch *`Bezeichner = Bezeichner op (Ausdruck)`* geschrieben werden. Also ist zum Beispiel `x div= 17` äquivalent zu `x = x div 17`.

Arithmetische Operatoren: Binäre arithmetische Operatoren erfordern genau zwei Operanden, zu beiden Seiten des Operators. Diese sind `+`, `-`, `*`, `/`, `div` (ganzzahlige Division), `%` (Divisionsrest), <<, >>, >>> (ein Bit nach rechts schieben ohne das Vorzeichenbit gesondert zu behandeln), `&`, `|` und `^`. Unäre arithmetische Operatoren benötigen einen Operanden, normalerweise rechts vom Operator (Ausgenommen Präinkrement und -dekrement). Die unären Operatoren sind `+`, `-` (Vorzeichenwechsel), `++` (Prä- oder Postinkrement), `-` (Prä- oder Postdekrement), und ∼ (bitweise Negation).

Logische Operatoren: An logischen Operatoren stehen `&&`, `||` und `!` für Und-Verknüpfung, Oder-Verknüpfung und Negation zur Verfügung. Bei den binären Operatoren `&&` und `||` wird der zweite Operand nur dann ausgewertet, wenn das Ergebnis nach Auswertung des ersten Operanden noch nicht feststeht (*lazy evaluation*). Das lässt sich nutzen, um mögliche Laufzeitfehler abzufangen: Beispielsweise wird im Ausdruck `y != 0 && x / y > 3` die Division nur dann durchgeführt, wenn `y` ungleich Null ist.

Vergleichsoperatoren: Die Vergleichsoperatoren sind <, `<=`, `==`, `>=`, > und `!=` mit den Bedeutungen kleiner, kleiner oder gleich, gleich, größer oder gleich, größer und ungleich. Die Vergleichsoperatoren liefern boolesche Werte. Die eingebauten Datentypen sind geordnet, sodass diese Operatoren funktionieren können. Für boolesche Werte gilt im Speziellen, dass `false` kleiner als `true` ist. Außerdem gilt noch: Ist einer der beiden Operanden `invalid`, so ist das Ergebnis des Vergleichs ebenfalls `invalid`, daher muss der `isvalid`-Operator verwendet werden, um herauszufinden, ob ein Ausdruck einen gültigen Wert besitzt oder nicht.

Komma-Operator: Der Komma-Operator (`,`) ist ein binärer Operator, der es gestattet, zwei Ausdrücke zu einem einzigen zusammenzufassen. Das Ergebnis der Operation ist das Ergebnis des zweiten Operanden.

Conditional-Operator: WMLScript besitzt wie C und Java den `?:`-Operator, der drei Operanden besitzt. Das Ergebnis des Gesamtausdruckes ist das Ergebnis des zweiten Operanden, wenn die Auswertung des ersten Operanden `true` ergibt, ansonsten ist es das Ergebnis des dritten Operanden. Folgendes Beispiel liefert den Maximumwert zweier Variablen: `x > y ? x : y`

`typeof`-**Operator:** Dieser unäre Operator ist eine Besonderheit von WMLScript. Er liefert eine Integer-Kodierung des Typs seines Operanden. Die Abbildung ist Integer auf `0`, Floating-point auf `1`, String auf `2`, Boolean auf `3` und Invalid auf `4`.

`isvalid`-**Operator:** Dieser Operator prüft, ob sein Operand nicht vom Typ Invalid ist (d.h. nicht den Wert `invalid` hat) und liefert einen booleschen Wert.

String Operatoren: Strings können mit den Operatoren `+` und `+=` Operator konkateniert werden. Für mächtigere Stringmanipulationen steht die String-Bibliothek zur Verfügung (siehe 4.9).

Array Operatoren: WMLScript unterstützt keine Arrays, wie man sie von anderen Programmiersprachen kennt. Stattdessen können Funktionen der String-Bibliothek (`elementAt`, `removeAt`, u. dgl.) verwendet werden, um auf einzelne, durch frei wählbare Zeichen getrennte, Elemente eines Strings zugreifen zu können.

Für eine Tabelle, die den Vorrang der Operatoren und ihre Assoziativität zeigt, sei auf [15, Seite 26ff.] verwiesen.

4.6. Variablen und Datentypen

Variablen werden mit dem Schlüsselwort `var` deklariert und können dabei auch initialisiert werden. Der Typ einer Variablen kann nicht festgelegt werden. Der Typ ergibt sich aus dem zugewiesenen Wert und kann sich auch zur Laufzeit des Programms ändern. WMLScript ist also eine schwach typisierte Sprache (*weakly typed language*). Beispiele für gültige Variablendeklarationen sind:

```
// Die einfachste Deklaration einer Variablen
var x;

// Mehrere Variablen können auf einmal deklariert werden
var a, b, c;

// Zusätzliche Initialisierung ist möglich
var i=17, j=true, k="ka";
```

Erfolgt keine explizite Initialisierung, wird die Variable automatisch mit dem Wert `""`, also einem leeren String, initialisiert.

Der Gültigkeitsbereich (*Scope*) einer Variablen erstreckt sich vom Ort ihrer Deklaration bis zum Ende der Funktion. Variablennamen müssen innerhalb einer Funktion eindeutig sein. Ein Block von Anweisungen stellt im Gegensatz zu anderen Programmiersprachen keinen eigenen Gültigkeitsbereich dar, das heißt, in einem Block kann keine Variable deklariert werden, wenn bereits eine gleichnamige Variable in einem äußeren Block existiert. Das impliziert auch, dass Variablen nicht durch andere verdeckt werden können, wie das zum Beispiel in der Programmiersprache C++ der Fall ist.

Diese Regel erlaubt es auch, Variablen im `if`-Zweig einer If-Anweisung zu deklarieren und im `else`-Zweig zu verwenden, was in anderen Sprachen nicht möglich ist. Beispiel:

```
if (success == true)
    var antwort = "richtig";
else
    antwort = "flahsc";
```

Die Lebensdauer einer Variablen reicht bis zum Ende der Ausführung der Funktion, in der sie deklariert ist.

Wie schon gesagt wurde, können Variablen Werte verschiedener Typen haben und der Typ kann sich auch ändern. Die Typen sind:

Integer: für ganze, vorzeichenbehaftete Zahlen mit 32 Bit Genauigkeit: Die größte (2147483647) und kleinste (-2147483648) als Integer darstellbare Zahl kann mit den Bibliotheksfunktionen `Integer.maxInt` und `Integer.minInt` ermittelt werden.

Floating-point: Das sind IEEE[3] Fließkommazahlen mit einfacher Präzision und 32 Bit[4] Auflösung. Die größte positive ($3{,}40282347 * 10^{38}$) und kleinste positive ($1{,}17549435 * 10^{-38}$) als Float darstellbare Zahl lässt sich mit den Bibliotheksfunktionen `Float.maxFloat()` und `Float.minFloat()` ermitteln.

String: Zeichenketten, die verglichen, konkateniert und manipuliert werden können

[3] „Eye-triple-E“, *Institute of Electrical and Electronics Engineers, Inc.* Siehe im Internet: `http://www.ieee.org`.

[4] 1 Vorzeichenbit, 8 Bit Exponent und 23 Bit Mantisse.

Boolean: boolesche Werte (`true` und `false`)

Invalid: Dieser Typ kann nur einen einzigen Wert haben, nämlich `invalid`. Variablen können vom Programmierer explizit auf `invalid` gesetzt werden, der Wert `invalid` kann aber auch aus einer ungültigen Berechnung resultieren, beispielsweise einer Division durch Null oder einer Inkompatibilität von Operanden in einer Operation.

Die Literale dieser Typen wurden bereits im Unterabschnitt 4.1 beschrieben.

Der Typ einer Variablen kann zur Laufzeit mit dem `typeof`-Operator ermittelt werden. Mit dem `isvalid`-Operator kann außerdem geprüft werden, ob der Operand ungleich `invalid` ist, was mit den Vergleichsoperatoren `!=` und `==` nicht möglich ist (siehe auch Unterabschnitt 4.5).

4.7. Automatische Typumwandlung

Der Typ einer Variablen wird bei der Programmierung in WMLScript nicht deklariert. Zur Laufzeit haben die Variablen aber trotzdem verschiedene Typen, wie im letzten Unterabschnitt 4.6 dargelegt wurde. Dadurch kann es passieren, dass die Bedeutung eines Operators erst zur Laufzeit des Programms festgelegt wird. Der Operator `+` beispielsweise kann sowohl die Bedeutung der arithmetischen Addition als auch die der Stringkonkatenation haben, je nachdem, ob der Typ seiner Operanden zur Laufzeit *Integer* oder *String* ist.

Außerdem kann es natürlich vorkommen, dass die beiden Typen der Operanden eines binären Operators nicht die gleichen sind. Dann versucht die Sprache einen der beiden Typen in den anderen zu konvertieren, um die Operation sinnvoll durchführen zu können.

Bei der Anwendung des `+`-Operators auf einen String und einen Integer gäbe es beispielsweise zwei Möglichkeiten: Die Sprache könnte den Integer in einen String konvertieren und die beiden Strings aneinander hängen oder sie könnte den String in einen Integer konvertieren und eine Addition der beiden Zahlen durchführen. Das Ergebnis der Operation wäre im einen Fall ein String und im anderen Fall ein Integer. Da bei diesem Beispiel die Konvertierung von String nach Integer nur dann möglich ist, wenn der String eine ganze Zahl repräsentiert, wird in diesem Fall von WMLScript die erste Variante durchgeführt.

Im Allgemeinen kann man sagen, dass WMLScript bei unterschiedlichen Typen versucht, eine Umwandlung in den jeweils mächtigeren Typ vorzunehmen, das heißt in den Typ, der die größere Wertemenge besitzt. Dies gilt allerdings nur unter der Voraussetzung, dass der Operator für den mächtigeren Typ definiert ist. Viele Operatoren erlauben überhaupt nur einen einzigen Typ für ihre oder ihren Operanden; in dem Fall muss dann unbedingt in diesen Typen konvertiert werden. Als Beispiel sei hier der >>-Operator (bitweise Verschiebung nach rechts) erwähnt, bei dem beide Operanden vom Typ Integer sein müssen.

Ist die Umwandlung eines Typen in einen anderen nicht möglich, wird in den Typ Invalid bzw. in den zugehörigen Wert `invalid` konvertiert. Das Ergebnis der jeweiligen Operation ist dann ebenfalls `invalid`, und das bedeutet, dass die Operation fehlgeschlagen ist bzw. der Ausdruck nicht berechnet werden konnte (siehe auch nächster Unterabschnitt 4.8).

Die Tabelle 4.1 zeigt die möglichen automatischen Typumwandlungen in WMLScript.

	Verwendet als Typ			
Gegeben	**Boolean**	**Integer**	**Float**	**String**
Boolean `true`	–	`1`	`1.0`	`"true"`
Boolean `false`	–	`0`	`0.0`	`"false"`
Integer `0`	`false`	–	`0.0`	`"0"`
Integer x <> `0`	`true`	–	Fließkommawert von x	Dezimaldarstellung von x
Float `0.0`	`false`	Illegal	–	`"0.0"`
Float x <> `0`	`true`	Illegal	–	Fließkommadarstellung von x
Leerstring `""`	`false`	Illegal	Illegal	–
String x <> `""`	`true`	Wert von x oder Illegal	Wert von x oder Illegal	–
`invalid`	Illegal	Illegal	Illegal	Illegal

Tabelle 4.1.: Typumwandlung in WMLScript (Quelle: [15, Kapitel 7.1.6])

Für genauere Informationen zur Typumwandlung, etwa eine Auflistung aller Operatoren inklusive der Typen ihrer Operanden und ihres Ergebnistyps, sei auf [15, Kapitel 7] verwiesen.

4.8. Laufzeit Fehlererkennung und -behandlung

WMLScript unterstützt keine Fehlerbehandlung wie Java oder C++, bei der Ausnahmen (*exceptions*)[5] beim Auftreten eines Fehlers geworfen werden und an anderer Stelle im Programm aufgefangen werden können.

Die Fehlerbehandlung von WMLScript ist wesentlich simpler. Es wird zwischen zwei Arten von Fehlern unterschieden: zum einen *fatale Fehler*, deren Eintreten die weitere Ausführung des Programms unmöglich machen, zum anderen *nicht-fatale Fehler*, die eine Programmfortsetzung und eine programmatische Fehlerbehandlung gestatten.

Bei einem fatalen Fehler wird das Programm abgebrochen und der User Agent (und damit hoffentlich auch der Benutzer) über diesen Umstand informiert. Ein nicht-fataler Fehler führt gewöhnlich dazu, dass das Ergebnis des fehlerhaften Ausdrucks den Wert `invalid` erhält.[6] Dieser besondere Wert kann dann mit dem `isvalid`-Operator (vgl. auch Unterabschnitt 4.5) überprüft und speziell behandelt werden.

Für eine vollständige und detaillierte Auflistung aller fatalen und nicht-fatalen Fehler sei auf [15, Kapitel 13.3 und 13.4] verwiesen.

Die fatalen Fehler lassen sich in aller Kürze folgenderweise gruppieren:

Bytecode Fehler: Zu dieser Gruppe gehören Fehler wie ungültige Argumentanzahl beim Aufruf einer Funktion, die Nichtauffindbarkeit einer aufgerufenen Funktion, Zugriffsverletzungen und dergleichen.

Programmierter Programmabbruch: Der Programmierer kann die Programmausführung mit der Bibliotheksfunktion `Lang.abort()` abbrechen.

Speichererschöpfung: Dazu gehören der Stapelüberlauf bzw. *Stack overflow* und der *Out of memory* Fehler.

Externe Ausnahmen: Der Benutzer oder das System können die Programmausführung zu jeder Zeit abbrechen.

[5] Ausnahmen sind in den beiden angesprochenen objektorientierten Programmiersprachen als Objekte der jeweiligen Sprache implementiert.

[6] Eine Ausnahme von dieser Regel ist der Unterlauf einer Fließkommazahl. Wenn eine Berechnung oder Typumwandlung einen Wert ergibt, dessen Betrag kleiner als `Float.minValue()` ist, wird das Ergebnis auf den Wert `0.0` gesetzt. Diese Art von Fehler kann also nicht auf einfache Weise erkannt werden.

Die nicht-fatalen Fehler sind:

Berechnungsfehler: Das sind die Division durch Null, der Integer oder Floating-point Überlauf und der Floating-point Unterlauf.

Fehler bei der Referenzierung von Konstanten: Dies betrifft Fließkomma-Literale, die einen der Werte *Not a Number*, *Positive Infinity* oder *Negative Infinity* (in IEEE754 definiert) repräsentieren. In diesen Fällen ist das Ergebnis `invalid`. Das Gleiche gilt für die Referenzierung von beliebigen Fließkomma-Literalen in einer reinen Integer-Umgebung.[7]

Umwandlungsfehler: Diese Fehler treten auf, wenn ein Wert bei der Umwandlung in einen Integer oder Float außerhalb des Wertebereichs dieser Typen liegt.

4.9. WMLScript-Standardbibliotheken

Wie im Unterabschnitt 4.3 beschrieben wurde, stellt WML einen Satz von Standardbibliotheken zur Verfügung, welche Funktionen beinhalten, die vom Programmierer verwendet werden können um Standardaufgaben zu lösen. Der Aufruf einer Bibliotheksfunktion hat die Form *Bibliotheksname*`.`*Funktionsname*`(`*Parameterliste*`)`, also etwa `Lang.min(x, y)` oder `String.charAt(str, idx)`.

WMLScript stellt die sechs Bibliotheken `Lang`, `Float`, `String`, `URL`, `WMLBrowser` und `Dialogs` zur Verfügung, die im Folgenden kurz beschrieben werden. Eine ausführlichere und vollständige Beschreibung der Standardbibliotheken findet sich in [16].

Die Lang-Bibliothek: Diese Bibliothek enthält Funktionen, die sich auf Sprachfeatures von WMLScript und Möglichkeiten des WMLScript-Interpreters beziehen. Dazu gehören beispielsweise `exit()` zur Beendigung der Programmausführung oder `random()` zum Generieren einer Pseudozufallszahl. Weiters stehen noch Funktionen zum Parsen von Zahlen aus Strings und einiges andere zur Verfügung.

[7] WMLScript darf auch auf Geräten laufen, die den Datentyp Float nicht unterstützen, siehe [15, Kapitel 14]. Ob Fließkommazahlen unterstützt werden, kann zur Laufzeit durch Aufruf der Bibliotheksfunktion `Lang.float()` erkannt werden.

Die Float-Bibliothek: Hier finden Funktionen zur Manipulation von Fließkommazahlen ihren Platz. `ceil()` und `floor()` zum Auf- und Abrunden gehören ebenso hierher wie `pow()` und `sqrt()` zur Berechnung von Potenzen und Wurzeln.

Die String-Bibliothek: Sie beinhaltet Funktionen zur Stringmanipulation, wie zum Beispiel `subString()` zum Ausschneiden eines Teilstrings. Aber auch Funktionen, mit denen sich Strings wie Arrays behandeln lassen, finden sich hier — dazu gehören etwa `elementAt()`, `insertAt()` oder `removeAt()`.

Die URL-Bibliothek: Hier finden sich Funktionen, mit denen eine URL in ihre Bestandteile *Schema*, *Host*, *Port*, *Pfad*, *Parameter*, *Query* und *Fragment* zerlegt werden kann. Außerdem können Strings mit `escapeString()` in eine Form gebracht werden, in der sie als Teil einer URL verwendet werden können.

Die WMLBrowser-Bibliothek: Diese Bibliothek beinhaltet Funktionen zur Interaktion und Steuerung des WML-Browsers. Mit `getVar()` und `setVar()` können WML-Variablen innerhalb von WMLScript manipuliert werden. Funktionen wie `go()` oder `prev()` haben die gleiche Bedeutung wie die entsprechenden Tasks in WML.

Die Dialogs-Bibliothek: Hier befinden sich drei Funktionen, durch die mit dem Benutzer interagiert werden kann. Diese sind `alert()` zur Anzeige von Nachrichten, `confirm()` zur Auswahl einer von zwei gegebenen Möglichkeiten und `prompt()` zur Eingabe von Text.

5. Wireless Telephony Application (WTA) und Wireless Telephony Application Interface (WTAI)

Die *Wireless Telephony Application* (WTA) ist ein Teil der WAP-Spezifikation, der die mobile Datenübertragung mit der mobilen Telefonie verbindet. WTA ermöglicht es beispielsweise, eine Telefonnummer aus einer WML-Card oder einem WMLScript heraus anzurufen. Der weitaus größere und komplexere Teil von WTA ist allerdings den Betreibern von Mobilfunknetzen vorbehalten und bietet diesen die Möglichkeit, die Mobiltelefone zu programmieren und dem Benutzer dadurch zusätzliche oder speziellere Dienste anzubieten.

Der Vorteil von WTA für die Mobilfunknetzbetreiber ist, dass sie WML und WMLScript verwenden können, um das Verhalten WAP-tauglicher Endgeräte verschiedener Hersteller einheitlich an ihre Bedürfnisse anzupassen. Das soll an folgendem Beispiel verdeutlicht werden: Normalerweise bestimmt die eingebaute Software des Handys, was mit einem eingehenden Anruf passieren soll — das heißt, in welcher Form er dem Benutzer präsentiert wird und welche Möglichkeiten dem Benutzer gegeben werden, auf ihn zu reagieren. Oft sind die gegebenen Möglichkeiten nur „Anruf entgegennehmen“ oder „Anruf ablehnen“. Mittels WTA kann der Mobilfunkbetreiber dieses Verhalten verändern. Er kann die Form der Präsentation (durch die Anzeige einer eigenen WML-Card) bestimmen und die Möglichkeiten des Benutzers (durch den Aufruf von WTAI-Funktionen aus WML oder WMLScript) erweitern — beispielsweise durch einen zusätzlichen Auswahlpunkt „An die Mobilbox weiterleiten“.

5.1. Die WTA-Architektur

Voraussetzung für die angesprochene Programmiermöglichkeiten ist, dass ein so genannter *WTA User Agent* am Mobiltelefon läuft (was nicht automatisch bei jedem WAP-Gerät der Fall ist). Dieser WTA User Agent muss in der Lage sein, WML und WMLScript darstellen bzw. abarbeiten zu können, er basiert also auf einem WML User Agent. Darüber hinaus steht er mit einem vom Netzbetreiber zur Verfügung

gestellten *WTA-Server* in Verbindung, der ihn mit so genannten *Services* versorgen kann. Diese Services können an *Events* gebunden werden, bei deren Eintreten sie ausgeführt werden. Die Services werden vom WTA User Agent im *Repository* persistent abgespeichert, um sie nicht jedes Mal vom WTA-Server holen zu müssen, wenn sie benötigt werden. Das Repository dient also als Cache für Services, welche auch ein Verfallsdatum haben. Die WTA-Architektur und der WTA User Agent sind in der Abbildung 5.1 dargestellt.

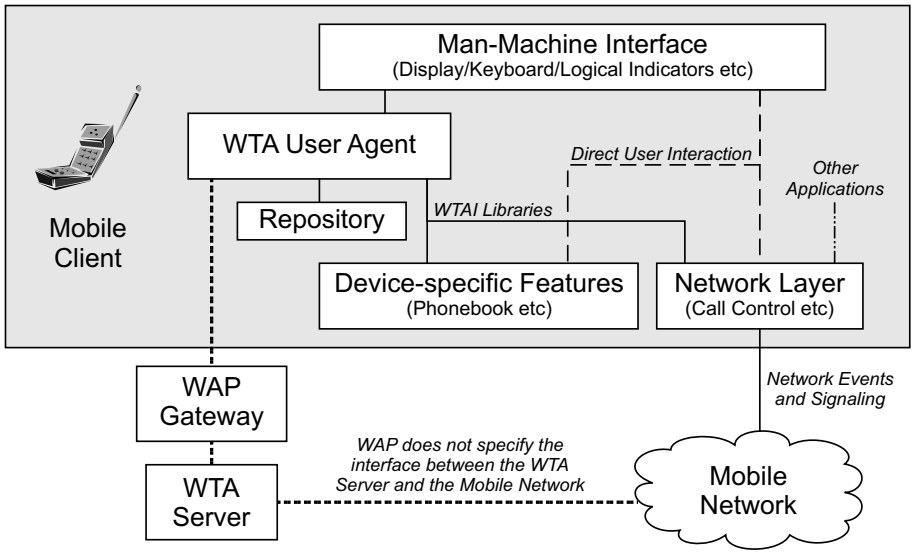

Abbildung 5.1.: Übersicht über die WTA-Architektur (aus [17, Figure 1])

Das Repository speichert so genannte *Channels*. Ein Channel definiert eine Menge von zusammengehörigen *Resourcen*, welche die Daten für einen Service beinhalten. Eine Resource kann dabei ein WML-Deck, ein WMLScript oder ein WBMP Bild sein. Das Laden eines Channels ist eine atomare Operation, das heißt, dass entweder alle zum Channel gehörenden Resourcen in das Repository geladen werden oder gar keine. Wie die Abbildung 5.2 auf der nächsten Seite zeigt, sind Überschneidungen erlaubt — eine Resource kann also von mehreren Channels gleichzeitig verwendet werden.

Technisch gesehen ist ein Channel ein XML-Dokument, das neben einem Namen, einem *Event Identifier* und anderen Informationen eine Liste von URIs auf die ihm zugehörigen Resourcen beinhaltet. Über den *Event Identifier* kann der Channel bzw. der durch ihn verkörperte Service an einen WTAI-Event gebunden werden. Beim Eintreten des Events wird das erste Element der Resourcenliste geladen und

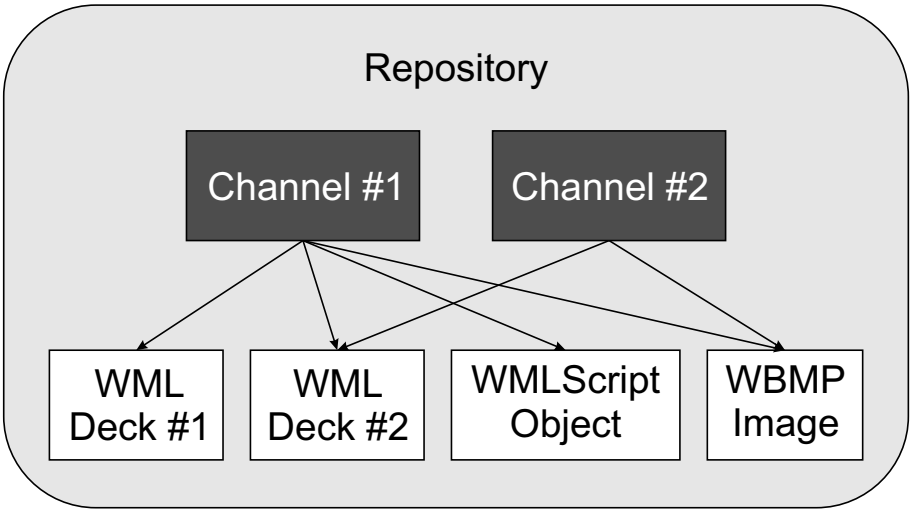

Abbildung 5.2.: Das Repository (aus [17, Figure 6])

ausgeführt. Dieses erste Element wird also normalerweise ein WML-Deck sein, das die anderen Resourcen im Bedarfsfall nachlädt.

Im Rahmen dieser Arbeit ist die Erstellung von Telefonieanwendungen durch Mobilfunknetzbetreiber in der beschriebenen Weise nur von sekundärem Interesse. Deshalb sei für detailliertere Informationen zur WTA-Architektur, dem Respository, Channels und dergleichen auf [17] verwiesen.

5.2. Sonderfall WAE User Agent

Wenn kein WTA User Agent zur Verfügung steht, können unter Umständen trotzdem Teile von WTA verwendet werden, wie die Abbildung 5.3 auf der nächsten Seite zeigt. In diesem Fall kommt ein *WAE User Agent* zum Einsatz, welcher nicht WTA-fähig ist. Üblicherweise ist das ein WML User Agent bzw. ein Microbrowser. Dieser WAE User Agent verfügt über kein Repository und kann auch keine Services an Events binden. Er kann aber einen Teil des *Wireless Telephony Application Interface*, die *Public WTAI*-Bibliothek, verwenden und mit deren Hilfe beispielsweise einen Telefonanruf aus einem WML-Deck heraus veranlassen. Auch diese Minimalfunktionalität von WTA nicht in alle derzeit im Handel erhältlichen WAP-Mobiltelefone integriert.

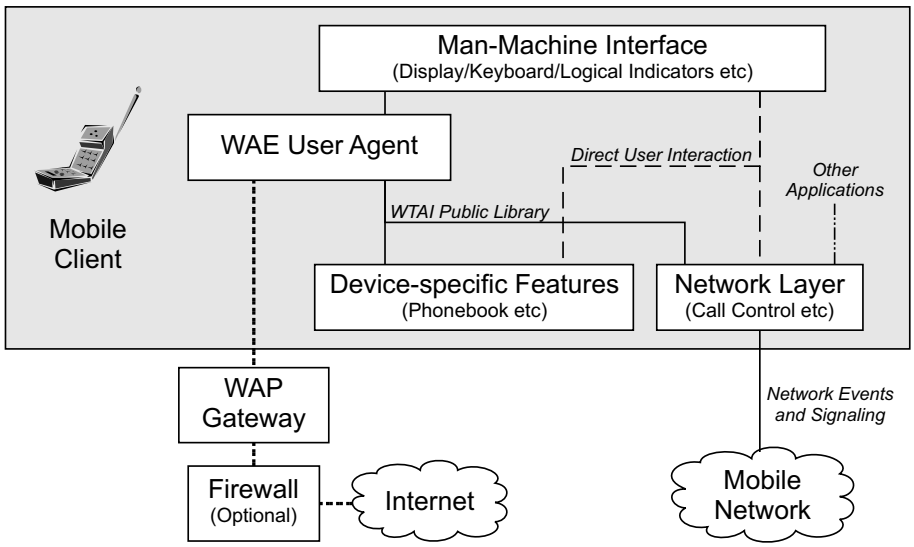

Abbildung 5.3.: WAE User Agent und WTAI Public Library (aus [17, Figure 2])

5.3. Das Wireless Telephony Application Interface

Das *Wireless Telephony Application Interface* (WTAI) ermöglicht die Erstellung von Anwendungen für die Telefonie. WTAI definiert dazu einen Satz von Bibliotheken, die aus WMLScript und teilweise auch aus WML heraus aufgerufen werden können. Außerdem spezifiziert und benennt das WTAI die Events, die (wie bereits angesprochen) an Services gebunden werden können.

Die WTAI-Bibliotheken sind in drei Bereiche unterteilt:

Network Common WTAI: Dieser Bereich deckt die Funktionalität ab, die den verschiedenen (von WTAI unterstützten) Mobilfunknetzen gemeinsam ist. Dazu gehört die Annahme eines eingehenden Telefonanrufes ebenso wie beispielsweise der Zugriff auf das integrierte Telefonbuch oder das Versenden und Empfangen von Textnachrichten.

Network Specific WTAI: Hier findet sich die Funktionalität, die auf eine bestimmte Netzwerktechnologie bezogen ist. Es gibt jeweils eigene Spezifikationen für das hauptsächlich in Europa verwendete GSM (*Global System for Mobile Communications*), das japanische PDC (*Personal Digital Cellular*) oder das neben anderen System in den USA verwendete IS-136 (*TDMA Cellular/PCS – Radio Interface – Mobile Station – Base Station Compatibility Standard*).

Public WTAI: In diesem Bereich sind einfache Funktionen untergebracht, die auch ohne einen WTA User Agent und den zugehörigen Event-Handling-Mechanismus verwendet werden können.

Die beiden ersten Bereiche sind den Betreibern von Mobilfunknetzen vorbehalten und erfordern einen WTA User Agent (siehe 5.1 auf Seite 45).

Die einzelnen Bereiche selbst bestehen wiederum aus einer oder mehreren Bibliotheken. Die Beschreibung zu den einzelnen Bibliotheken ist in den Tabellen 5.1 und 5.2 aufgelistet.

Funktionsbibliothek	*Name*	*Beschreibung der Bibliothek*
Public WTAI	„wp“	allgemein verfügbare WTAI-Funktionen

Tabelle 5.1.: Public WTAI-Funktionsbibliotheken

Funktionsbibliothek	*Name*	*Beschreibung der Bibliothek*
Voice Call Control	„vc“	ermöglicht den Gesprächsaufbau und bietet die Kontrolle über das Gerät *während* eines Gesprächs (im Gegensatz zum Public WTAI)
Network Text	„nt“	bietet Funkionen zum Senden und Empfangen von Textnachrichten
Phonebook	„pb“	erlaubt den Zugriff auf das Telefonbuch des Gerätes
Call Logs	„cl“	gestattet den Zugriff auf die Gesprächsprotokolle des Gerätes
Miscellaneous	„ms“	beinhaltet diverse andere Funktionen

Tabelle 5.2.: Network Common WTAI-Funktionsbibliotheken

In der Spalte „Name“ der Tabellen stehen Abkürzungen der Bibliotheksbezeichnungen. Diese Abkürzungen müssen beim Aufruf einer WTAI-Bibliotheksfunktion aus WML angegeben werden, um die jeweilige Bibliothek zu benennen. Die einzelnen Funktionen einer Bibliothek haben zu diesem Zweck ebenfalls Abkürzungen mit zwei oder mehr Buchstaben.

Der Aufruf einer WTAI-Funktion aus WML geschieht über ein eigenes URI-Schema. Dadurch können die WTAI-Funktionen an den meisten Stellen aufgerufen werden, an denen in WML ein URI erlaubt ist, beispielsweise im `href`-Attribut

des `go`-Elementes oder im `ontimer`-Attribut des `card`-ELementes. Das URI-Schema hat die Form `wtai://`*`Bibliothek/Funktion`* `(; `*`Parameter`*`)* [! `*`Ergebnis`*`]`, wobei *Bibliothek* und *Funktion* die angesprochenen Identifikationskürzel sind, *Parameter* den Wert eines Parameters darstellt und *Ergebnis* eine WML-Variable benennt, in der der Rückgabewert der Funktion abgelegt wird.[1] Der WTAI-URI „`wtai://wp/sd;314159265!res`“ würde beispielsweise die Public WTAI-Funktion „Send DTMF Tones“ aufrufen und ihr den Wert „314159265“ übergeben. Der Wert der WML-Variable `res` würde anschließend den Rückgabewert der WTAI-Funktion widerspiegeln.

Eine zweite Aufrufmöglichkeit besteht aus WMLScript heraus. Hier werden die WTAI-Bibliotheken als zusätzliche Bibliotheken der Programmiersprache (siehe 4.3 auf Seite 33) zur Verfügung gestellt und können in der gewohnten Weise verwendet werden. Die Anweisung `var res = WTAPublic.sendDTMF("314159265");` wäre also das WMLScript-Äquivalent zum oben angeführten URI-Beispiel. Man sieht, dass hier die Benennung der Bibliothek und der Funktion etwas ausführlicher ist.

Wie bereits in 5.1 auf Seite 45 erklärt wurde, sind die den Mobilfunknetzbetreibern vorbehaltenen Möglichkeiten zur Erstellung von Telefonieanwendungen im Rahmen dieser Arbeit nur von untergeordnetem Interesse. Deshalb wird hier auf die einzelnen Funktionen der Network Common WTAI und der Network Specific WTAI nicht mehr näher eingegangen. Es sei stattdessen auf [18] verwiesen, wo auch die verschiedenen Arten von Events beschrieben werden.

5.3.1. Public WTAI

Die Public WTAI-Bibliothek enthält drei Funktionen, die auch in Anwendungen verwendet werden können, die nicht von einem WTA-Server stammen. Die Existenz eines WTA User Agents am Endgerät ist also für ihre Verwendung nicht erforderlich. Sie können in einem WML-Microbrowser zur Verfügung stehen und in WML oder WMLScript aufgerufen werden. Sinn macht das natürlich nur, wenn das Gerät ein Mobiltelefon ist und somit in der Lage ist, einen Telefonanruf durchzuführen.

[1] Die Leerzeichen dienen zur besseren Lesbarkeit und werden im URI nicht angegeben. Mit „`( ... )*`“ wird gekennzeichnet, dass der eingeschlossene Teil gar nicht oder beliebig oft hintereinander angegeben werden darf. Die Zeichenfolge „`[ ... ]`“ bedeutet, dass der eingeschlossene Teil optional ist.

Ein wesentlicher Unterschied zu anderen WTAI-Bibliotheken und deren Funktionen ist, dass dem Benutzer die Möglichkeit gegeben sein muss, die Ausführung einer Public WTAI-Funktion zu verhindern.

Die Bibliothek hat die Abkürzung „wp“ für den Aufruf einer ihrer Funktionen aus einem WML-Deck. Der Name der zugehörigen WMLScript-Bibliothek lautet „WTAPublic“. Die drei Funktionen sind:

Make Call

Diese Funktion dient dazu, den Anruf der angegebenen Telefonnummer zu veranlassen. Die anzurufende Nummer muss allerdings am Display des Gerätes angezeigt werden, und der Anruf muss vom Benutzer bestätigt werden.

URI:	`wtai://wp/mc;`*`Nummer`*`[!`*`Ergebnis`*`]`
WMLScript:	`makeCall(`*`Nummer`*`);`
Parameter:	*Nummer* (String): Das ist die Telefonnummer, die angerufen werden soll. Es dürfen die für eine Telefonnummer gültigen Zeichen verwendet werden (also „0“–„9“ und „+“).
Ergebnis:	*Ergebnis* (Integer): Null, wenn der Anruf erfolgreich durchgeführt werden konnten oder eine negative Zahl im Fehlerfall
Beispiel:	URI: `wtai://wp/mc;%2B4331612345` WMLScript: `WTAPublic.makeCall("+4331612345");`

Send DTMF Tones

Sendet MFV-Töne[2] über eine aufrechte Gesprächsverbindung. Damit können Kommandos zur Interaktion mit einem automatisierten Sprachausgabesystem übertragen werden.

URI:	`wtai://wp/sd;`*`DTMF`*`[!`*`Ergebnis`*`]`
WMLScript:	`sendDTMF(`*`DTMF`*`);`

[2]MFV steht für *Mehrfrequenzwahlverfahren*, bei dem Wähltöne durch die Überlagerung zweier diskreter Frequenzen erzeugt werden. Es wird auch oft einfacher als *Tonwahlverfahren* bezeichnet. Die englische Bezeichnung lautet *Dual Tone Multiplexed Frequency* mit der Abkürzung DTMF.

Parameter:	*DTMF* (String): eine Sequenz von gültigen Tonwahl-Zeichen („0“–„9“, „A“–„D“, „#“, „*“ und „,“)
Ergebnis:	*Ergebnis* (Integer): Null, wenn die Tonwahl-Zeichen erfolgreich gesendet werden konnten oder eine negative Zahl im Fehlerfall
Beispiel:	URI: `wtai://wp/sd;*1234` WMLScript: `WTAPublic.sendDTMF("*1234");`

Add Phonebook Entry

Mit dieser Funktion kann ein Paar, bestehend aus einer Telefonnummer und aus der mit ihr assoziierten Bezeichnung, zum Telefonbuch des Gerätes hinzugefügt werden. Der Benutzer muss die Ausführung auch hier bestätigen, nachdem ihm die Nummer und die Bezeichnung angezeigt wurden.

URI:	`wtai://wp/ap;`*Nummer*`;`*Name*`[!`*Ergebnis*`]`
WMLScript:	`addPBEntry(`*Nummer*`, `*Name*`);`
Parameter:	*Nummer* (String): die Telefonnummer, die gespeichert werden soll *Name* (String): der Name, der mit der Telefonnummer assoziert werden soll
Ergebnis:	*Ergebnis* (Integer): Null im Erfolgsfall oder eine negative Zahl im Fehlerfall
Beispiel:	URI: `wtai://wp/ap;031612345;Pizza` WMLScript: `WTAPublic.addPBEntry("031612345", "Pizza");`

Ein Beispiel zu Public WTAI

Das abschließende Beispiel zeigt, wie die Public WTAI-Funktion „Make Call“ aus einer WML-Card aufgerufen wird.

```
<?xml version="1.0" encoding="ISO-8859-1"?>
<!DOCTYPE wml PUBLIC "-//WAPFORUM//DTD WML 1.2//EN"
                     "http://www.wapforum.org/DTD/wml_1.2.xml">
<wml>
  <card>
    <p>
```

```
      Was wollen Sie bestellen?

      <select>
        <optgroup>
          <option onpick="wtai://wp/mc;%2B4331612345">
            Italienisch
          </option>

          <option onpick="wtai://wp/mc;%2B4331654321">
            Chinesisch
          </option>
        </optgroup>
      </select>
    </p>
  </card>
</wml>
```

Für den Aufruf wird hier das `onpick`-Attribut des `option`-Elementes verwendet (vgl. Abschnitt 2.2 auf Seite 19). Alternativ wäre natürlich auch die Verwendung des `go`-Tasks (vgl. 2.2.2 auf Seite 20) möglich gewesen. Innerhalb des URI wird die Zeichenfolge „`%2B`“ verwendet, um das `+`-Symbol der Telefonnummer darzustellen.

Teil II.

Mobiler Zugang zum Hyperwave IS/6

6. Der Hyperwave IS/6

Der Hyperwave IS/6 ist ein System, mit dem verschiedenste Dokumente verwaltet werden können, also ein *Dokument Management System*. Die Dokumente werden zusammen mit Meta-Informationen in einer Datenbank abgelegt und können hierarchisch strukturiert werden. Zwischen den Dokumenten können Bezüge in Form von Hyperlinks existieren, deren Konsistenz vom System automatisch verwaltet wird. In der Hyperwave-Terminologie heißen die Dokumente auch *Objekte* und die Meta-Informationen sind die *Attribute* der Objekte.

Das System kann mit mehreren Clients angesprochen werden, von denen die wichtigsten kurz beschrieben werden:

- Der Hyperwave IS/6 kann ein Web-Interface generieren, das den Zugriff mit einem WWW-Browser gestattet. Das ist die wichtigste Schnittstelle zum System.
- Die *Hyperwave Virtual Folders* integrieren sich (ähnlich wie die Netzwerkumgebung) in das Dateisystem von Windows und gestatten so einen Zugriff auf die im Hyperwave IS/6 abgelegten Dokumente mit dem Windows Explorer.
- Es existiert ein Satz von Kommandozeilen-Tools, die unter anderem das Importieren und Exportieren von Dokumenten ermöglichen.
- Durch ein eigenes *Application Programming Interface* (API) wird die Einbindung von Hyperwave in andere Systeme ermöglicht.

Die zentralen Features des Hyperwave IS/6 sind:

Automatic Link Management: Die Hyperlinks werden von den Dokumenten getrennt gespeichert und erst beim Betrachten eines Dokumentes wieder an den richtigen Stellen eingemischt. Dadurch können Dokumente verschoben werden, und die Links, die sich auf sie beziehen, bleiben trotzdem funktionsfähig. Innerhalb von Hyperwave kann es daher auch keine „toten“ Hyperlinks geben —

wenn ein Dokument gelöscht wird, werden Links auf dieses Dokument nicht mehr angezeigt.

Multi-User-Umgebung: Es können Benutzer und Benutzerrechte definiert werden. Der Zugriff auf Dokumente kann eingeschränkt werden, und es sind sogar benutzerabhängige Sichten auf ein Dokument möglich.

Suchfunktionen: Eine schnelle Volltextsuche ist ebenso möglich wie die Suche nach Schlüsselwörtern in den Attributen der Dokumente.

Eltern-Kind-Beziehungen: Dokumente können hierarchisch organisiert werden, wobei ein Dokument auch mehrere Eltern haben kann. Die hierarchische Struktur wird durch ein eigenes Container-Dokument, der so genannten *Collection* und ihrer Varianten, ermöglicht.

Dokumentklassen: Das System gestattet die Definition von Dokumentklassen, die das Aussehen und Verhalten einer Klasse von zusammengehörigen Dokumenten programmatisch beschreiben. Dokumentklassen können dann instanziert werden — es können also Dokumente bzw. Objekte am Server angelegt werden, die entsprechend ihrer Klasse visualisiert werden. Dadurch wird beispielsweise auch die Gruppierung mehrerer Objekte zu einer logischen Einheit ermöglicht.

Neben den Möglichkeiten des Document Management (der *Hyperwave eKnowledge Suite* (EKS)) gestattet das System auch die Entwicklung und Ausführung anderer Applikationen, wie zum Beispiel der *Hyperwave eLearning Suite* (ELS), einem Web Based Training-System, oder einer Portaloberfläche, dem *Hyperwave eKnowledge Portal* (EKP).

Hyperwave verwendet zur Generierung des Web-Interfaces und zur Ausführung von Dokumentklassen serverseitiges JavaScript. Die Code-Fragmente in diesem Teil der Arbeit sind daher häufig in dieser Sprache verfasst.

7. HTML und WML auf einem Server

Um sowohl Web-Browser als auch WAP-Geräte simultan bedienen zu können, muss abhängig vom Client entweder HTML- oder WML-Output generiert werden. Dafür gibt es grundsätzlich zwei Möglichkeiten:

- Es werden zwei WaveMaster-Prozesse mit unterschiedlichen Templates konfiguriert. Einer bedient Web-Browser mit HTML, der andere WAP-Geräte mit WML. Die Trennung wird dann durch unterschiedliche Adressierung erreicht — die beiden WaveMaster-Prozesse können zum Beispiel verschiedene Hostnamen haben oder verschiedene Portnummern belegen.

- Die andere Möglichkeit ist, dass lediglich ein einziger WaveMaster-Prozess verwendet wird. Dieser muss dann aber in der Lage sein, zwischen den verschiedenen Typen von Clients zu unterscheiden. Hier lässt sich nochmals zwischen zwei Varianten unterscheiden:

  - Es kann auch hier anhand der Adressierung unterschieden werden. Beispielsweise könnte ein URL-Präfix (wie etwa „`/wap/`“) verwendet werden, um im Template die Generierung von WML zu veranlassen.
  - Das Template enthält Code um den Client-Typ erkennen zu können.

Die zuletzt genannte Möglichkeit ist die interessanteste, da sie mit einem einzigen Wavemaster-Prozess auskommt und keine geänderte Adressierung erfordert. Der Benutzer kann also Adressen, die ihm vom Web-Zugang her bekannt sind, mit dem WAP-Gerät unverändert übernehmen. Außerdem ermöglicht dieser Ansatz die Entwicklung eines Template-Moduls, das sich in das Komponenten-Konzept von Hyperwave integrieren lässt, wie das Kapitel 9 auf Seite 64 zeigt.

Um zu erkennen, welche Art von Client bedient werden soll, kann der `Accept`-Header von HTTP verwendet werden (vgl. [3]). Der User Agent gibt in diesem Feld an, welche MIME Media Types (oder ganze Bereiche von MIME Media Types) er verarbeiten kann. Ein WAP User Agent im Speziellen muss den MIME Type

„`text/vnd.wap.wml`“ akzeptieren. Wenn der User Agent diesen Mime Type nicht akzeptiert, handelt es sich also sicher nicht um einen WML-Browser, und es muss HTML generiert werden.

Der Umkehrschluss ist nicht möglich. Die gängigen Web-Browser akzeptieren den MIME Type „`*/*`“, sind also in der Lage beliebige Mime Types zu verarbeiten oder zumindest abzuspeichern. Einen möglichen Ausweg aus dieser Situation bietet der `User-Agent`-Header von HTTP, welchen jeder User Agent setzen sollte. Mit seiner Hilfe kann für bestimmte bekannte Browser auf jeden Fall HTML generiert werden, auch wenn sie „`text/vnd.wap.wml`“ akzeptieren. Dies ist speziell für Hyperwave ein guter Ansatz, da das Produkt ohnehin nur bestimmte Web-Browser unterstützt.

Die folgende JavaScript-Funktion funktioniert nach dem beschriebenen Prinzip, wobei der Einfachheit halber MIME Type Bereiche wie „`*/*`“ nicht geprüft werden:[1]

```
function isWapClient() {
  function acceptsMimeType(token) {
    var x = request.field["Accept"].split(/\s*[,;]\s*/);
    for (var i in x)
      if (x[i] == token)
        return true;
    return false;
  }

  if (!acceptsMimeType("text/vnd.wap.wml"))
    return false;

  return request.agent.name != "MSIE" &&
         request.agent.name != "Netscape";
}
```

Wenn diese Funktion am Beginn des Templates aufgerufen wird, kann anschließend die jeweils richtige Ausgabe generiert werden. Natürlich kann die Funktion beliebig komplex erweitert werden, wenn es die Umstände erfordern; beispielsweise durch die Verwendung einer Konfigurationsdatei, in der die Eigenschaften der User Agents eingetragen werden können. WAP 1.2 definiert außerdem so genannte *User Agent Profiles*, die auch für diesen Zweck verwendet werden können. Diese User Agent Profiles können als Parameter beim `Accept`-Header angegeben werden und sind in [19] definiert.

[1] Die Variable `request.agent` ist ein String, der vom Server aus dem `User-Agent`-Header berechnet wird.

8. Längenbeschränkung der Inhalte

WML-Browser haben üblicherweise ein Limit, was die Dokumentgröße betrifft. Diese Grenze wird durch den vergleichsweise geringen Speicher dieser Geräte bestimmt. Außerdem existieren in den meisten Fällen keine Möglichkeiten, RAM-Speicher im Bedarfsfall auf ein anderes Medium auszulagern, wie man das von Personal Computern kennt. Aus den genannten Gründen können mobile Endgeräte oft nur wenige Kilobytes pro Server-Response verarbeiten, und die darstellbaren Dokumente sind auf diese Größe beschränkt.

Dadurch ergibt sich eine Situation, die sich wesentlich von der im World Wide Web unterscheidet. Heutige Web-Browser können mehr oder weniger beliebig große Dokumente verarbeiten. Zumindest stößt man nur selten an die Grenzen — HTML Dokumente beispielsweise können durchaus mehrere hundert Kilobytes groß sein, verschachtelte Tabellen oder andere komplexe Strukturen beinhalten und lassen sich trotzdem noch relativ problemlos darstellen.

Eine WAP-Komponente für Hyperwave muss diesen wesentlichen Unterschied berücksichtigen. Zum einen können in Hyperwave gespeicherte Dokumente sehr groß sein und zum anderen kann auch die zur Navigation generierte Ausgabe beliebig groß werden. Beispielsweise hängt die Größe eines Collection Listings von der Anzahl der Kinder der Collection ab — und Collections mit vielen hundert Kindern sind durchaus im Bereich des Normalen.

Aus den beschriebenen Einschränkungen ergeben sich zwei Notwendigkeiten:

1. Am Server abgelegte Dokumente, die zu groß für den Client sind, können nicht unverändert an ihn gesendet werden. Mehrere Möglichkeiten bieten sich an:

   - Dem Benutzer kann eine Meldung angezeigt werden, dass das Dokument auf seinem Gerät nicht angezeigt werden kann.
   - Das Dokument kann auf die Maximalgröße des Clients gekürzt werden, was aber mit einer entsprechenden Mitteilung an den Benutzer verbunden sein sollte.

- Eine Aufteilung in mehrere kleinere Dokumente kann möglich sein. Für die Navigation kann dann am Ende eines Dokumentes ein Hyperlink auf das nächste Dokument in der Liste erstellt werden.

Welche der angesprochenen Möglichkeiten die geeignetste ist, hängt auch vom Dokumenttyp ab. Für eine Textdatei ist die Aufteilung in kleinere Dokumente (mit gleichzeitiger Konvertierung nach WML) relativ unproblematisch. Andere Dokumente können unter Umständen gar nicht oder nur sehr schwer übermittelt werden, man denke etwa an Multimediadateien. Auch XML eignet sich nicht gut zur Aufteilung, da keine Tags aufgebrochen werden dürfen und die Document Type Definition nicht missachtet werden darf.

2. Dynamisch generierte Inhalte dürfen die Maximalgröße ebenfalls nicht überschreiten. Bei langen Listen (Collection Listings, Suchergebnissen oder dergleichen) muss also die Möglichkeit zum Blättern gegeben sein.

Die maximale Größe eines Dokumentes ist selbstverständlich geräteabhängig. Die Dateigrößen können für bekannte WML-Browser abhängig vom `User-Agent`-Header in einer Tabelle abgelegt werden. Ist das Endgerät WAP 1.2-tauglich, kann zusätzlich auch auf das im Kapitel 7 auf Seite 57 angesprochene *User Agent Profile* zurückgegriffen werden. Dieses Profil beinhaltet ein Feld namens `WmlDeckSize`, das die maximale Größe eines WML-Decks in Bytes angibt.

8.1. Schätzung der Bytecode-Größe

Zwischen dem Server und dem Client bzw. WML-Browser befindet sich normalerweise das WAP-Gateway (vgl. Kapitel 1 auf Seite 5), das WML von der Klartextform in eine binäre Darstellung, den so genannten *Bytecode*, komprimiert. Die maximale Größe eines WML-Decks, die der Client downloaden kann, bezieht sich natürlich auf die komprimierte Form der Daten.

Der Server kann bei der Generierung von Listen den einfachen und sicheren Weg gehen und diese Komprimierung durch das Gateway ignorieren. Wenn er den Kompressionsfaktor nicht kennt, muss er vom schlimmsten Fall ausgehen, nämlich dass die Daten gar nicht komprimiert werden können. Er muss also dann das Größenlimit des Clients auf die Klartextdarstellung von WML beziehen. Bedenkt man aber, dass

die tatsächliche Komprimierung relativ hoch sein kann, würde so unter Umständen sehr viel verfügbarer Platz ungenutzt bleiben.

Um die verfügbare Größe bestmöglich nutzen zu können, muss der Server also bereits beim Generieren eines WML-Decks die Information besitzen, wie groß das Deck nach der Konvertierung in den Bytecode sein wird. Da er dies aber nicht exakt bestimmen kann, ohne die Konvertierung tatsächlich durchzuführen, ist es hilfreich, die Bytecode-Größe zu schätzen.

Eine solche Schätzung sollte nach folgenden Kriterien vorgenommen werden:

- Sie muss schnell durchführbar sein. Sie muss auf jeden Fall wesentlich schneller sein als die tatsächliche Konvertierung in den Bytecode.
- Sie sollte für kurze Textblöcke funktionieren, aus denen die Ausgabe zusammengebaut wird. Es sollte also zum Beispiel die Größe eines einzelnen Listenelementes abschätzbar sein, sodass entschieden werden kann, ob dieses einzelne Element noch in der Liste Platz findet.
- Die Schätzung muss nicht unbedingt exakt sein. Wenn sie allerdings falsche Werte für die Byteanzahl liefert oder liefern kann, müssen diese auf jeden Fall über dem tatsächlichen Wert liegen, sodass die maximale Größe nicht überschritten werden kann. Unter Umständen kann dabei Geschwindigkeit gegen Exaktheit eingetauscht werden.

Die WML-Spezifikation ([11]) beschreibt im Kapitel 14 die Binärrepräsentation von WML. Mit Hilfe der dortigen Angaben lassen sich einige Regeln angeben, mit denen sich die Bytecode-Größe ziemlich genau abschätzen lässt:

- Die Tags eines WML-Dokumentes (das sind <a>, <anchor>, ..., <wml>) werden durch jeweils ein Byte dargestellt.
- Hat ein Tag einen Inhalt (also Text und/oder andere Tags) kommt noch ein Byte pro Tag hinzu, um das Ende des Inhalts anzuzeigen.
- Besitzt ein Tag Attribute wird ein weiteres Byte benötigt, um das Ende der Attributliste zu kennzeichnen.

- Text zwischen Tags wird als String abgespeichert, wobei es zwei Möglichkeiten gibt. Sie können *inline* oder in einer referenzierbaren Tabelle gespeichert werden. Im schlimmsten Fall wird ein String bei jedem Auftreten inline gespeichert und benötigt dann zwei Bytes mehr, als seine Länge ist.

- Eine Folge von *white space character* innerhalb von Textpassagen wird als einzelnes Leerzeichen kodiert, benötigt also nur ein einziges Byte.

- Bestimmte *character entities* (` `, `&quot;`, `&#123;` oder dergleichen) benötigen im ungünstigsten Fall 4 Bytes.

- Attribute von Tags werden als Paare von Namen und Werten gespeichert. Die erlaubten Attributnamen (das sind `accept-charset`, `access`, ..., `xml:lang`) werden durch jeweils ein einziges Byte dargestellt.

- Attributwerte werden wie Text gespeichert (also wie beschrieben zwei Bytes plus Stringlänge und Zusammenfassung von *white space characters* zu einem einzelnen Leerzeichen), jedoch gibt es für häufig vorkommende Werte (wie z. B. `onenterforward`) und Wertteile (wie etwa `http://www.`) wieder Abkürzungen, die nur ein einziges Byte benötigen.

Die Hyperwave WAP-Komponente beinhaltet eine in JavaScript implementierte Schätzfunktion mit dem Namen „`estimatedBytecodeSize()`“, die auf den gemachten Angaben basiert. Die Funktion trennt den Eingabestring zunächst in eine Liste von Tokens. Ein solches Token repräsentiert entweder Text oder ein (öffnendes oder schließendes) Tag. Die Bytecode-Größe der einzelnen Tokens wird dann gemäß obiger Regeln aufsummiert.

8.2. Ausgabe von Listen in mehreren Teilen

Bei der Navigation durch die Dokumente eines Hyperwave Servers müssen oftmals Listen verschiedener Art erstellt werden. Dazu gehören Listen von Kindern einer Collection, Listen von Tabs in einem Desktop eines Portals und andere. Man steht also in verschiedenen Situationen vor dem gleichen Problem, nämlich eine Liste zu generieren, die die maximale Deckgröße des Clients so gut wie möglich nützt und (falls notwendig) am Ende einen Link auf die Fortsetzung einfügt.

Zu diesem Zweck existiert in der WAP-Komponente von Hyperwave eine eigene Funktion namens `buildWmlList()`, die Listen durch Hinzufügen einzelner Einträge erzeugen kann, deren Bytecode-Größe aufaddiert wird, bis die maximale Größe des Decks oder eine vorgegebene maximale Elementanzahl erreicht wird.

Die exakte Implementierung dieser Funktion ist für diese Arbeit nicht weiter von Interesse und wird daher nicht wiedergegeben — da sie aber von manchen in den folgenden Kapiteln angegebenen Code-Beispielen verwendet wird, sei kurz auf auf ihre Funktionsweise eingegangen:

Vor dem Aufruf von `buildWmlList()` wird üblicherweise ermittelt, was vor und nach der Liste stehen soll. Dazu gehören die Struktur der Decks und der Cards, Menüs, Text und andere Inhalte. Wenn diese Daten bekannt sind, kann ihre Bytecode-Größe ermittelt werden. Die Differenz aus maximaler Deckgröße und dem bereits vergebenen Platz ergibt die Größe, die maximal für die Liste verbleibt. Dieser Wert wird der Funktion übergeben. Ein weiterer wichtiger Parameter ist eine Funktion, die einzelne Elemente der Liste liefert, also ein *Callback*, der intern von `buildWmlList()` aufgerufen wird.

Andere Parameter von `buildWmlList()` legen fest, bei welchem Element der Liste begonnen werden soll oder wie viele Elemente maximal ausgegeben werden sollen. Wenn das letzte Element der Liste nicht mehr Platz findet, erzeugt die Funktion automatisch einen Link auf die Fortsetzung. Dazu können noch Funktionsparameter angegeben werden, die festlegen, welche Form der Link haben soll und welcher Text dabei ausgegeben werden soll (z. B. „Weiter...“ oder „Mehr...“).

9. Die Hyperwave WAP-Komponente

Der Standard-Templatesatz von Hyperwave beinhaltet ein Komponentenkonzept. Komponenten sind eigenständige Module, die für bestimmte Aufgaben zuständig sind. Es gibt beispielsweise Komponenten für die Versionskontrolle von Dokumenten, für spezielle Features für Administratoren (Anlegen von Benutzern und Gruppen und dgl.) oder auch Hilfsfunktionen für das Portal.

Einer der Vorteile dieser Modularisierung der Funktionalität ist, dass nicht benötigte Komponenten auf einfache Weise deaktiviert werden können, indem sie nicht inkludiert werden.

Um zu verstehen, wie Komponenten in das System eingebunden werden können, muss zunächst erklärt werden, was man im Hyperwave-Kontext unter *Actions* versteht. Mit den Actions kann das Verhalten der Standard-Templates gesteuert werden. Die Templates können damit zum Beispiel veranlasst werden, ein Dialogfenster zu öffnen, das die Eingabe bestimmter Daten ermöglicht. Ein anderes Beispiel ist eine Action, die bewirkt, dass ein Dokument ohne das sonst übliche HTML-Frameset dargestellt wird. Actions haben einen Namen (zum Beispiel „`insert.user.action`“) und werden ausgelöst, indem sie als Teil der Request-URL angegeben werden. Der erste Parameter einer Action ist üblicherweise das Hyperwave-Objekt, das in der URL angegeben ist — Actions beziehen sich also auf dieses Objekt und bestimmen, was mit ihm geschehen soll. Weitere Parameter können in der URL nach der Action angegeben werden.

Die Verbindung zwischen Actions und Komponenten wird mit dem so genannten *Actionhandler* hergestellt. Komponenten können sich bei diesem Actionhandler registrieren. Die Registrierung bewirkt, dass sie bei der Generierung der Ausgabe berücksichtigt werden. Der Actionhandler hat drei Funktionen, die im Rahmen dieser Arbeit von Interesse sind („`actionhandler`“ ist dabei eine Instanz der WMLScript-Klasse „`hw_Class_actionhandler`“):

actionhandler.component(mname, mstring): Diese Funktion dient zur Registrierung von Komponenten. Der erste Parameter ist dabei entweder der Name ei-

nes PLACE-Makros[1] oder eine JavaScript-Funktion. Im Falle einer JavaScript-Funktion muss im zweiten Parameter noch ein Name für die Komponente angegeben werden. Der Actionhandler trägt die zu registrierende Komponente in eine interne Liste ein, um sie später beim Aufruf von `dispatch()` verwenden zu können. Komponenten werden zu Beginn der Abarbeitung von `master.html` registriert.

actionhandler.dispatch(): Diese Funktion wird im Template `master.html` genau einmal aufgerufen, um die in der URL angegebene Action zu starten. Dazu wird die Liste der registrierten Komponenten durchlaufen und der Reihe nach jede mittels `actionhandler.component()` übergebene JavaScript-Funktion aufgerufen oder das alternativ mögliche PLACE-Makro evaluiert. Jeder aufgerufenen Komponente steht es dann frei, Output zu generieren. Wenn eine Komponente in diesem Sinn aktiv wird, kann sie den Actionhandler veranlassen, alle weiteren Komponenten außer Acht zu lassen, indem sie `actionhandler.status()` aufruft.

actionhandler.status(): Diese Funktion kann von Komponenten aufgerufen werden, um den weiteren Ablauf von `actionhandler.dispatch()` zu steuern. Normalerweise übergibt eine Komponente den Wert `"done"`, nachdem sie Output generiert hat, um die Abarbeitung der Komponentenliste zu beenden

Die einzelnen Komponenten sollen bei ihrem Aufruf durch `actionhandler.dispatch()` anhand der in der URL angegebenen Action überprüfen, ob sie eine Aufgabe zu erledigen haben. Wenn keine ihnen bekannte Action angegeben ist, erzeugen sie keinen Output, und die nächste Komponente kommt an die Reihe. Der folgende Code-Ausschnitt zeigt die übliche Funktion einer Komponente:

```
actionhandler.component(menu_dcmaster, "menu_dcmaster");

function menu_dcmaster() {
  actionhandler.status("done");

  var action = request.act;
  if (action.indexOf("dc.defaultdialog") == 0) {
```

[1] PLACE ist eine von Hyperwave selbst entwickelte Makro-Sprache, die mit einfachen Konstrukten die Generierung von dynamischen Seiten ermöglicht. Sie kann serverseitig parallel zu JavaScript verwendet werden (JavaScript wurde erst nach PLACE in Hyperwave integriert).

```
    resdef("v6.0/components/menu_dc/langres/menu_dc.lrc", "menu");
    hw_retobj = hw_dc_createDefaultDialogFromURL();
  }
  else if (action.indexOf("dc.execute") == 0) {
    resdef("v6.0/components/menu_dc/langres/menu_dc.lrc", "menu");
    hw_retobj = hw_dc_executeMethodFromUrl();
  }
  // ...
  // Weiter Abfragen mit action.indexOf()...
  // ...
  else
    actionhandler.status("");
}
```

In diesem Beispiel wird also im Dispatching die JavaScript-Funktion `hw_dc_createDefaultDialogFromURL()` aufgerufen, wenn die in der URL angegebene Action mit „`dc.defaultdialog`“ beginnt, und so weiter.

Man sieht also, dass dieses bestehende Komponentenkonzept für die Entwicklung einer WAP-Komponente nicht optimal geeignet ist. Wenn der Client ein WML-Browser ist, soll schließlich WML generiert und die Ausgabe von HTML gänzlich unterbunden werden. Die WAP-Komponente soll also exklusive Kontrolle über die Ausgabe erhalten und nicht nur bestimmte Actions für sich reservieren. Und auch wenn gar keine Action angegeben ist, soll WML ausgegeben werden.

Dies ist mit einem Kunstgriff möglich, der allerdings der bisher beschriebenen „Philosophie“ der Komponenten widerspricht. Die WAP-Komponente kann an Stelle von Actions, die im Kapitel 7 auf Seite 57 beschriebene Funktion `isWapClient()` verwenden, um zu entscheiden, ob sie aktiv werden soll. Wenn die Komponente aktiv wird und WML generiert, kann sie dem Actionhandler mitteilen, dass keine anderen Komponenten mehr verwendet werden sollen, indem sie `actionhandler.status("stopProcessing")` aufruft.[2] Die Komponente kann aber trotzdem den Action-Mechanismus verwenden, um spezielle Aufgaben zu erfüllen, zum Beispiel die Konvertierung von Textdokumenten nach WML.

Das folgende Codefragment zeigt den Aufbau der WAP-Komponente. Das Beispiel ist stark vereinfacht und dient nur zur Veranschaulichung der bisherigen Aussagen. Der tatsächlich im Einsatz befindliche Code setzt neben „`Content-Type`“ noch die HTTP Response-Header „`Expires`“ und „`Cache-Control`“, bestimmt die Sprache

[2] „`stopProcessing`“ wird an Stelle von „`done`“ verwendet, damit der nach den Komponenten aufgerufene Errorhandler ebenfalls keine Ausgaben mehr produziert.

des Clients aus dem HTTP Request-Header „Accept-Language“ und kennt und behandelt mehrere Actions.

```
if (isWAPClient())
  actionhandler.component(wapmaster, "wapmaster");

function wapmaster() {
  actionhandler.status("stopProcessing");

  response.field["Content-Type"] = "text/vnd.wap.wml";

  if (wmlIdentify())
    return;

  switch (request.act) {
    case "hw_wap_image.action":
      var obj = server.object({objectidentifier:request.actp});
      if (obj.error.error()) {
        hw_wap_error = obj.error;
        break;
      }
      wmlImageOutput(obj.object);
      break;

    case "hw_wap_plaintext.action":
      var obj = server.object({objectidentifier:request.actp});
      if (obj.error.error()) {
        hw_wap_error = obj.error;
        break;
      }
      wmlPlainTextOutput(obj.object, "/;internal&amp;action=" +
        request.act + "&amp;Parameter=" + request.actp);
      break;

    default:
      wmlOutput();
      break;
  }

  wmlProcessError();
}
```

Die Komponente übernimmt die Kontrolle wenn `isWapClient()` `true` liefert. Abhängig von der Action wird jeweils eine eigene JavaScript-Funktion aufgerufen.[3] Wenn die URL keine Action beinhaltet, wird die Funktion `wmlOutput()` aufgerufen.

Die Funktion `wmlProcessError()` dient zur Behandlung von Fehlern, die entweder im Server selbst oder in der WAP-Komponente aufgetreten sind. Mit serverseitigen Fehlern sind zum Beispiel fehlende Benutzerrechte oder nicht existierende Objekte gemeint.

Bevor Output generiert wird, wird noch die Funktion `wmlIdentify()` aufgerufen. Sie sorgt dafür, dass der Benutzer am System angemeldet ist. Auf die Schwierigkeiten bei der Authentifizierung wird im folgenden Abschnitt eingegangen.

9.1. Die Anmeldung am System

Die Multi-User Umgebung ist eines der wichtigsten Features des Hyperwave IS/6. Das Web-Interface gestattet sowohl anonymen Zugang als auch die An- und Ummeldung des aktuellen Benutzers. Registrierte Benutzer können Gruppen zugeordnet werden und können verschiedene Rechte haben. Wichtig ist insbesonders, dass Dokumente nicht nur für bestimmte Benutzer zugänglich gemacht oder verborgen werden können, sondern dass sie auch benutzerabhängige Sichtweisen gestatten.

Als zu Beginn des Jahres 2000 mit der Entwicklung der WAP-Komponente für den Hyperwave IS/6 begonnen wurde, war nur ein einziges WAP-fähiges Handy in Österreich erhältlich. Dieses Handy unterstützte zwar *HTTP Basic Authentication*, hatte aber die unangenehme Eigenart sich das Benutzername-Passwort-Paar für jede besuchte URL zu merken. Eine Ummeldung und das damit verbundene permanente Wechseln des aktuellen Benutzers war also nicht möglich, da beim Besuchen einer bekannten Seite wieder unbemerkt zum alten Benutzer zurückgewechselt wurde.[4]

Aus diesem Grund konnte die Benutzeranmeldung über WAP nicht in der eigentlich gewünschten Weise analog zum Web-Interface implementiert werden. Die gewählte Realisierung unterstützt zwei Modi:

[3] Auf `wmlImageOutput()` wird im Abschnitt 9.6 auf Seite 75 eingegangen. Für `wmlPlainTextOutput()` sei auf Abschnitt 9.3 auf Seite 71 verwiesen.

[4] Dieses Verhalten zeigte sich auch dann, wenn bei der Anmeldung unterschiedliche *Basic Realms* verwendet wurden.

1. Ist die JavaScript-Variable `HW_Settings.WAP_USE_ANONYMOUS_ACCESS` in der Datei `config/defaults.js` auf `true` gesetzt, so ist **nur** anonymer Zugang zum System möglich.

2. Ist `HW_Settings.WAP_USE_ANONYMOUS_ACCESS` auf `false` gesetzt, so ist **kein** anonymer Zugang möglich. Stattdessen wird eine einmalige Anmeldung des Benutzers erzwungen — die Um- oder Abmeldung ist nicht möglich.

Im ersten Fall wird der Benutzer nie zur Identifizierung aufgefordert. Wenn er zu einem Objekt gelangen will, das für anonyme Benutzer nicht sichtbar ist, wird ihm eine entsprechende Fehlermeldung angezeigt.

Im zweiten Fall darf der Benutzer nicht anonym sein. Wenn er also anonym zu einem Objekt gelangt wird eine Seite generiert, die ihn zur Anmeldung auffordert. Dabei wird immer der gleiche *Basic Realm* gesetzt, was den WML User Agent dazu veranlasst, die zuletzt für diesen Realm verwendeten Benutzerinformationen zu übertragen. Dadurch wird der Benutzer nur einmal zur Eingabe dieser Daten aufgefordert.

Die von der WAP-Komponente aufgerufene JavaScript-Funktion wmlIdentify() sieht folgendermaßen aus:

```
function wmlIdentify() {
  if (!wapAnonymousAccess())
    if (server.user().object.uid == 'anonymous') {
      response.code = "401";
      response.reason = "Access denied";
      response.field["WWW-Authenticate"] = 'Basic realm="Hyperwave"';

      writeWmlErrorDeck("Error", "Authentication required");

      return true;
    }

  return false;
}
```

Die Funktion `wapAnonymousAccess()` beinhaltet die Abfrage der zuvor angesprochenen Variablen `HW_Settings.WAP_USE_ANONYMOUS_ACCESS`. Mit `writeWmlErrorDeck()` wird ein einfaches WML-Deck mit einer Fehlermeldung erstellt.

9.2. Die Ausgabe von Objekten ohne spezielle Actions

Wenn man nochmals die Funktion `wapmaster()` betrachtet, sieht man, dass nach der Identifizierung die Actions behandelt werden. Wenn die URL keine Action beinhaltet, wird schließlich die Funktion `wmlOutput()` aufgerufen. Aufgabe dieser Funktion ist es, das angeforderte Objekt auf die *übliche Weise* darzustellen. Damit ist gemeint, dass das Objekt gemäß seinem Typ an den Client gesendet wird — bei einer Collection wird die Liste der Kinder angezeigt, Plain-Text wird in WML konvertiert, WML-Bilder und WMLScripts werden unverändert übertragen. Mit den Actions kann dieses Verhalten verändert werden — so kann beispielsweise ein WMLScript nach WML konvertiert werden, um am Client des Sourcecode zu lesen, statt das Script auszuführen.

Die Funktion `wmlOutput()` muss also anhand des Objekttyps entscheiden, wie weiter zu verfahren ist. Der Objekttyp kann dabei zunächst nach Hyperwave-Objektart unterschieden werden, also Collection, Dokument, Instanz einer Dokumentklasse oder dergleichen. Im Falle eines Dokumentes muss außerdem noch sein MIME Media Type betrachtet werden. Die Funktion `wmlOutput()` wird im Folgenden vereinfacht wiedergegeben:

```
function wmlOutput() {
  var goid = request.object.getGOid();
  switch(request.object.MimeType) {
    case "text/plain":
      wmlPlainTextOutput(request.object, "/" + goid);
      break;

    case "text/html":
      wmlHtmlOutput(request.object, "/" + goid);
      break;

    default:
      hw_retobj = hw_dc_checkAndExecuteShowMethod();
      if (!hw_retobj.status)
        if (request.object.DocumentType == "collection")
          wmlCollectionListOutput(request.object);
        else
          hw_wap_error = new HW_API_Error('Error: Cannot display ' +
            'object of type ' + request.object.MimeType);
      break;
  }
}
```

Wie man sieht, wird zunächst nach MIME Type unterschieden (wobei davon ausgegangen wird, dass nur Hyperwave-Objekte vom Typ Dokument über das MimeType-Attribut verfügen). Wenn das Objekt über keinen oder einen unbekannten MIME Type verfügt, wird als Nächstes geprüft, ob es sich um eine Instanz einer Dokumentklasse handelt, die über eine eigene Show-Methode verfügt. Dies geschieht in der Funktion `hw_dc_checkAndExecuteShowMethod()`, die die Show-Methode auch gleich ausführt, wenn sie vorhanden ist.[5] Dieser Schritt ist für den drahtlosen Zugang zum *Hyperwave eKnowledge Portal* notwendig, da dieses auf Dokumentklassen basiert (siehe Kapitel 10 auf Seite 77).

Wenn die Show-Methode nicht ausgeführt werden konnte, wird zuletzt geprüft, ob das Objekt eine Collection ist. Wenn ja, wird die Liste ihrer Kinder ausgegeben. Im anderen Fall konnte das Objekt nicht klassifiziert werden, und es wird eine Fehlermeldung ausgegeben.

9.3. Die Ausgabe von Textdokumenten

Wie gezeigt wurde, wird die Funktion `wmlPlainTextOutput()` aufgerufen, wenn der MIME Type des Dokuments „`text/plain`“ ist oder wenn die Textausgabe explizit durch Angabe der Action „`hw_wap_plaintext.action`“ veranlasst wird.

Text kann nicht unverändert an einen WAP-Client geschickt werden, sondern muss zunächst in WML umgewandelt werden. Außerdem muss berücksichtigt werden, dass das Textdokument möglicherweise zu groß ist, um auf einmal dargestellt werden zu können.

Die Funktion `wmlPlainTextOutput()` übernimmt diese beiden Aufgaben und ist hier in einer etwas gekürzten Version (im Wesentlichen ohne Fehlerbehandlung) wiedergegeben:

```
function wmlPlainTextOutput(obj, uri) {
  var title = obj.title(client.language);
  var wml_head = beginWml() + beginCard("main", title) +
    wmlCommonMenu() + '<p>';
  var wml_tail = '</p>' + endCard() + wmlInfoCard(obj) + endWml();

  var used_size = estimatedBytecodeSize(wml_head) +
    estimatedBytecodeSize(wml_tail);
```

[5] `hw_dc_checkAndExecuteShowMethod()` ist nicht Teil der WAP-Komponente, sondern wird auch von den Standard-Templates zur Visualisierung von Dokumentklassen verwendet.

```
  var remaining_size = wapMaxDeckSize() - used_size;

  var content = server.content({objectidentifier:obj.getGOid(),
    mode:2}).content.read(-1);
  var text_array = content.replace(/\r\n/g, '\n').split('\n');

  function getLine(i) {
    return text_array[i].replace(/&/g, '&amp;')
      .replace(/</g, '&lt;').replace(/>/g, '&gt;')
      .replace(/̈/g, '&quot;').replace(/́/g, '&apos;') + '<br/>';
  }

  var start = getParametersFromUrl(request.url)['start'];
  var wml_main = buildWmlList({start:start, end:text_array.length-1,
    size:remaining_size, getLine:getLine, uri:uri});

  write(wml_head, wml_main, wml_tail);
}
```

Wie man sieht, wird die im Abschnitt 8.2 auf Seite 62 vorgestellte Funktion `buildWmlList()` für den Aufbau der Ausgabe verwendet, um durch den Inhalt des Textdokuments blättern zu können. Der Text wird also als Liste interpretiert, wobei die Listenelemente den einzelnen Zeilen des Texts entsprechen.

Das Dokument wird also vom Server gelesen und in seine einzelnen Zeilen zerlegt. Dieses Feld von Zeilen wird in der Variablen `text_array` abgelegt, die von der Funktion `getLine()` verwendet wird, um die einzelnen Zeilen an `buildWmlList()` zu liefern. Innerhalb von `getLine()` werden dann noch bestimmte Sonderzeichen wie spitze Klammern ersetzt, die in WML eine eigene Bedeutung haben und daher nicht direkt für den Inhalt verwendet werden können. Außerdem wird noch „
“ angehängt, um das Zeilenende in WML zu kennzeichnen.

Vor dem Aufruf von `buildWmlList()` wird der Rest des zu erstellenden WML-Decks bestimmt und der verbleibende Speicherplatz für die Liste ermittelt. Dazu werden kleinere Hilfsfunktionen wie `beginWml()`, `beginCard()` und dergleichen verwendet, die häufig benötigten WML-Code erzeugen und hier nicht näher erklärt werden sollen.

Die offensichtlichen Probleme der Funktion `wmlPlainTextOutput()` in der angegebenen Form sind, dass das Dokument nicht zu groß sein sollte, da das gesamte Dokument gelesen und im Speicher in seine einzelnen Zeilen zerlegt wird. Außerdem versagt die Listengenerierung, wenn eine einzelne Zeile länger als die maximale

Deckgröße des WAP-Clients ist, da diese Zeile dann auch auf der Fortsetzungsseite nicht Platz haben kann.

9.4. Die Ausgabe von HTML-Dokumenten

Die Ausgabe von HTML-Dokumenten ist noch wesentlich schwieriger als die von Textdokumenten. Dass HTML nicht Eins zu Eins in WML konvertiert werden kann ist klar — schließlich hat HTML wesentlich komplexere Strukturen (man denke nur an verschachtelte Tabellen) und HTML-Dokumente sind meistens für eine bestimmte Bildschirmauflösung entworfen, die am WML-Browser nicht zur Verfügung steht.

Dass die Syntax von WML (bzw. XML) wesentlich strenger ist als die von HTML, macht die Sache auch nicht einfacher. Wenn zum Beispiel in HTML zu einem -Tag das schließende -Tag fehlt, müsste dieses bei der Konvertierung nach WML an geeigneter Stelle eingefügt werden, um eine Fehlermeldung am Client zu vermeiden.

Eine andere Art von Problem kann auftreten, wenn zum Beispiel eine lange HTML-Tabelle (ohne Verschachtelung) nach WML konvertieren werden soll und die maximale Deckgröße berücksichtigt werden muss. Dann müsste die Tabelle nach einer Reihe geteilt und in zwei oder mehrere kleinere Tabellen zerlegt werden, also auf einer Seite schließende Tags eingefügt und auf der Fortsetzungsseite mit den entsprechenden öffnenden Tags fortgefahren werden. Wenn die maximale Größe bei einem Spaltenwechsel der Tabelle überschritten wird, ist nicht einmal das möglich.

Ziel einer dynamischen Konvertierung nach WML kann es also bestenfalls sein, nur den eigentlichen Text aus einem HTML-Dokument zu extrahieren, einfache Texthervorhebungen, Zeilenwechsel und Hyperlinks zu erhalten und diesen Text dann (wie im letzten Abschnitt beschrieben) nach WML zu konvertieren. Es können natürlich aufwendigere Konvertierungen vorgenommen werden, wenn bereits beim Upload eines HTML-Dokumentes eine WML-Version generiert und das Dokument am Server doppelt gespeichert wird.

Die Funktion `wmlHtmlOutput()` der WAP-Komponente begnügt sich damit, HTML nach WML zu konvertieren, indem die meisten Tags sowie Client-Side JavaScript-Funktion und Style-Sheet-Angaben aus dem Inhalt entfernt werden. Das Ergebnis wird dann wieder mittels `buildWmlList()` in einzelne Seiten zerlegt, durch die geblättert werden kann.

9.5. Collection Listings und Query Objects

Die Ausgabe von Collection Listings ist vergleichsweise einfach: Mittels der Funktion `server.children()` werden die Kinder der Collection vom Server geholt. Anschliessend wird die Liste der Kinder sortiert und kann mittels der bereits bekannten Hilfsfunktion `buildWmlList()` ausgegeben werden.

Die so genannten Query Objects können gemeinsam mit den Collections behandelt werden. Query Objects sind Hyperwave-Objekte, in denen Suchanfragen gespeichert sind. Sie verhalten sich wie Collections, deren Kinder dem Ergebnis der Suche entsprechen. Die Suche wird beim Öffnen eines Query Objects durchgeführt, sodass das Ergebnis immer auf dem aktuellen Stand ist. Query Objects werden als solche gekennzeichnet, indem das Attribut „`ContainerType`“ auf den Wert „`Query`“ gesetzt wird.

Bei der Ausgabe des Collection Listings sind Query Objects zu berücksichtigen, indem die Funktion `hw_savedQuery()` aufgerufen wird, die die Liste der Kinder mit dem Suchergebnis füllt.

Das folgende Codefragment zeigt die von `wmlOutput()` (vgl. Seite 70) aufgerufene Funktion `wmlCollectionListOutput()` in vereinfachter Form:

```
function wmlCollectionListOutput(obj) {
  var wml_head = beginWml() + beginCard("main",
    obj.title(client.language)) + wmlCommonMenu() + '<p>';
  var wml_tail = '</p>' + endCard() + wmlInfoCard(obj) + endWml();

  var used_size = estimatedBytecodeSize(wml_head) +
    estimatedBytecodeSize(wml_tail);
  var remaining_size = wapMaxDeckSize() - used_size;

  var children = server.children({objectidentifier:obj.getGOid()});
  var sortorder = '(DocumentType:E:collection)T';
  children.objects.sort(sortorder, client.language);

  if (obj.ContainerType == "Query")
    hw_savedQuery(obj, children.objects);

  if (children.objects.length == 0) {
    write(wml_head, "Empty collection.", wml_tail);
    return;
  }

  function getLine(i) {
```

```
    var entry = children.objects[i];
    return '<a href="' + getShortestObjectPath(entry) +
      '">' + entry.title(client.language) + '</a><br/>';
  }

  var start = getParametersFromUrl(request.url)['start'];
  var end = children.objects.length - 1;
  var wml_main = buildWmlList({start:start, end:end,
    size:remaining_size, getLine:getLine});

  write(wml_head, wml_main, wml_tail);
}
```

Es wird also eine Liste erstellt, deren einzelne Elemente den Kindern der Collection bzw. den Suchergebnissen entsprechen. Diese Listenelemente werden von der Funktion `getLine()` erstellt und zurückgeliefert. Aufgabe dieser Funktion ist es also, einen Link auf das jeweilige Kind bzw. das jeweilige Suchergebnis zu liefern, der vom Benutzer verfolgt werden kann, um zu diesem Objekt zu gelangen.

9.6. Die Ausgabe von WBMP-Bildern

Wenn ein WBMP-Bild mittels des `img`-Tags in eine WML-Seite eingebettet ist, bedarf es keiner speziellen Behandlung. Beim Einspielen in den Server wird das Bild auf Grund seiner Dateierweiterung bzw. seines MIME Media Types als solches erkannt und gekennzeichnet. Dazu wird das Objektattribut „`DocumentType`“ auf den Wert „`Image`“ gesetzt. Beim Anzeigen der WML-Seite wird dann das Bild automatisch vom Browser nachgeladen. Der Hyperwave IS/6 erkennt anhand des `DocumentType`-Attributes, dass es sich um ein Bild handelt, und sendet seinen Inhalt zusammen mit dem Mime Media Type an den Browser. Das bedeutet, dass beim Versenden von Bilddaten die Templates nicht zum Einsatz kommen, die Datei `master.html` also nicht abgearbeitet wird. Deshalb wird in der Funktion `wmlOutput()` (vgl. Seite 70) bei der Analyse des Mime Media Types die Bilddatentypen außer Acht gelassen.

Anders verhält es sich, wenn ein WBMP-Bild Teil einer Collection ist und der Benutzer das Bild über das Collection Listing anwählt. In diesem Fall darf nicht einfach ein Hyperlink auf das Bild erstellt werden, da manche WML-Browser ein WBMP-Bild nur dann darstellen können, wenn es in eine WML-Seite eingebettet ist. Es muss vielmehr eine WML-Seite dynamisch generiert werden, die ihrerseits das Bild nachlädt.

Zur Erzeugung dieser (bis auf das Bild) leeren WML-Seite wird die `hw_wap_image`-Action verwendet, auf die in der Funktion `wapmaster()` (vgl. Seite 67) geprüft wurde. Wenn diese Action gesetzt ist, wird die im Folgenden angegebene Funktion `wmlImageOutput()` zum Erzeugen der einbettenden WML-Seite aufgerufen:

```
function wmlImageOutput(obj) {
  write(beginWml(),
    beginCard("main", obj.title(client.language)),
    wmlCommonMenu(), '<p><img src="',
    getShortestObjectPath(obj), '" alt="[img]"/></p>',
    endCard(), wmlInfoCard(obj), endWml());
}
```

Nun muss noch die `hw_wap_image`-Action ausgelöst werden, wenn der Benutzer im Collection Listing ein Bild anwählt. Das Collection Listing selbst muss also im Hyperlink auf das Bild die Action beinhalten. Dazu muss die `getLine()`-Hilfsfunktion von `wmlCollectionListOutput()`, wie folgt, angepasst werden:

```
function getLine(i) {
  var entry = children.objects[i];
  var entry_title = entry.title(client.language);

  if (entry.DocumentType == "Image") {
    if (entry.MimeType == "image/vnd.wap.wbmp")
      return '<a title="Image" href="/;internal&amp;action=' +
        'hw_wap_image.action&amp;Parameter=' +
        entry.getGOid() + '">' + entry_title + '</a><br/>';
    else
      return entry_title + '<br/>';
  }

  return '<a href="' + getShortestObjectPath(entry) +
    '">' + entry_title + '</a><br/>';
}
```

Wie man sieht, wird für Bilder, die nicht vom Typ „`image/vnd.wap.wbmp`“ sind, kein Link erstellt, da sie am WML-Browser ohne Konvertierung nicht darstellbar sind.

10. WAP-Unterstützung für das Hyperwave eKnowledge Portal

Das Hyperwave eKnowledge Portal ist eine auf Dokumentklassen basierende Applikation, die es erlaubt, an beliebigen Stellen der Collection-Hierarchie ein Portal zu instanzieren. Ein solches Portal ist in Abbildung 10.1 wiedergegeben.

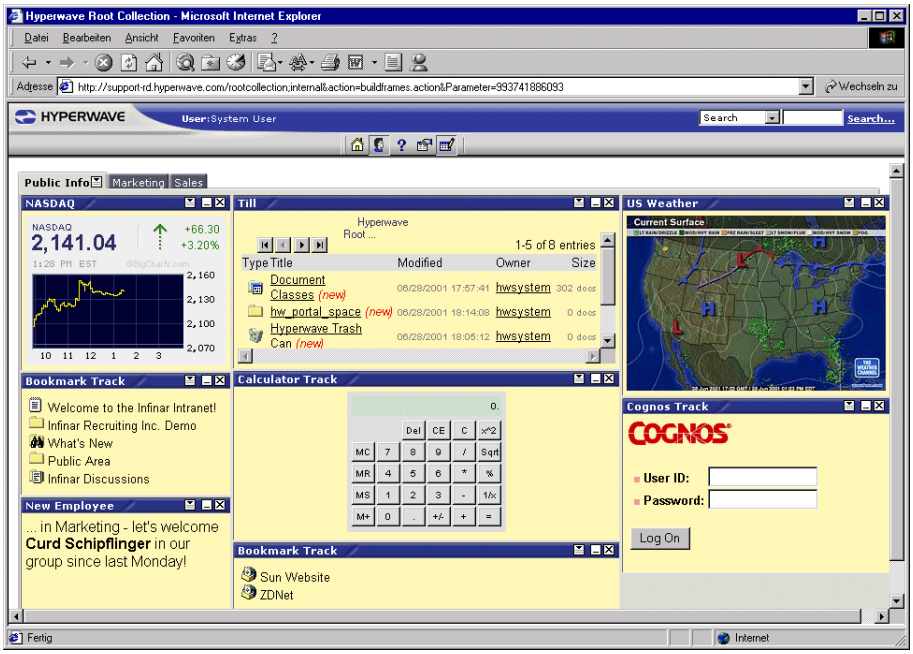

Abbildung 10.1.: Das Hyperwave eKnowledge Portal (http://www.hyperwave.com)

Die Oberfläche des Portals wird als *Desktop* bezeichnet. Die einzelnen Elemente des Portals sind die so genannten *Tracks*; in der Abbildung sind dies „US Weather“, „Calculator Track“ und so weiter. Das System erlaubt es dem Benutzer, sein persönliches Portal mit den für ihn interessanten Tracks zusammenzustellen. Dabei können die Tracks noch beliebig gruppiert werden. Für diesen Zweck stehen die so genannten *Tabs* zur Verfügung; das sind die Reiter „Public Info“, „Marketing“ und „Sales“ in der Abbildung.

Manche Tracks können (bzw. müssen) vom Benutzer konfiguriert werden. Beispielsweise ist der in der Abbildung gezeigte „US Weather“-Track tatsächlich ein Track, der beliebige Bilder aus dem World Wide Web laden und darstellen kann. Er wird mit der URL des Bildes und einem Aktualisierungsintervall konfiguriert. Diese Konfiguration kann im Web-Interface beim Anlegen des Tracks vorgenommen werden.

Ein Portal wird, wie folgt, am Server gespeichert:

- Der Desktop ist eine Instanz der Dokumentklasse `HW_PortalDesktop` und wird innerhalb der Collection angelegt, in der das Portal erscheinen soll.
- Ein Tab ist eine Instanz von `HW_PortalTab` und wird als Kind seines Desktops abgelegt. Ein Desktop muss mindestens einen Tab besitzen.
- Die Tracks werden als Kinder eines Tabs gespeichert. Ein Track ist eine Instanz einer Dokumentklasse, die von `HW_PortalTrack` abgeleitet ist, beispielsweise `HW_PortalTrackPicture` oder `HW_PortalTrackPeopleFinder`. Es bleibt den Tracks selbst überlassen, wie sie ihre Konfigurationseinstellungen ablegen. Die einfachste Variante ist, diese Daten als Hyperwave-Objektattribute der Instanz abzulegen. Die Tracks können aber auch Kinderobjekte erzeugen, da sie sich selbst wie Collections verhalten.

Bei der Anzeige des Portals im Web-Browser passiert in vereinfachter Form Folgendes:

1. Die URL des Desktops wird im Browser (in einem eigenen Frame) geladen.
2. Der Server erkennt, dass das angeforderte Objekt eine Instanz einer Dokumentklasse ist und ruft dessen `hw_gui_show()`-Methode auf, um zu erfahren, welche Methode beauftragt werden soll, um die Dokumentklasse zu visualisieren. Für die Darstellung im Web-Browser ergibt das die Methode `renderDesktopContent()`.
3. Nun wird `renderDesktopContent()` aufgerufen, um den HTML-Code für das Portal zu generieren. Diese Dokumentklassen-Methode ist, wie das ganze Portal, in JavaScript implementiert und kann durch Aufrufe von `write()` und `writeln()` Strings ausgeben, die dann an den Client geschickt und im Browser dargestellt werden.

4. `renderDesktopContent()` ermittelt mittels `api.children()` seine Tabs und fügt die so genannten Role-Tabs zu dieser Liste hinzu. Einer dieser Tabs ist der im Augenblick aktuelle Tab, also der, den der Benutzer betrachten will.

5. Als Nächstes wird das HTML-Grundgerüst erzeugt – also Tabellen, welche die Reiter der Tabs beinhalten, ein Rahmen für den aktuellen Tab, Menüs zum Bearbeiten der Tabs (mittels Client-Side JavaScript) und dergleichen mehr.

6. Die weitere Ausgabe wird an den aktuellen Tab delegiert, indem seine `renderTabContent()`-Methode aufgerufen wird.

7. Der aktuelle Tab arbeitet in analoger Weise weiter, indem er seine Kinder (also die darzustellenden Tracks) ermittelt, Rahmen für die Tracks erzeugt und Client-Side JavaScript für die Menüs der Tracks generiert.

8. Wenn ein Track sichtbar ist, wird seine `renderTrackContent()`-Methode aufgerufen, deren Aufgabe es ist, den HTML-Code des Tracks selbst auszugeben.

Dabei ist zu beachten, dass der Client für die Darstellung des Portals nur die URL des Desktops kennt und nur dieses eine Objekt vom Server anfordert. Die gesamte Darstellung des Portals wird vom Desktop gesteuert, und Tab- oder Track-Instanzen werden vom Client bzw. Web-Browser nie direkt angefordert. Der folgende Abschnitt zeigt, dass dies für den drahtlosen Zugang anders gelöst wurde.

10.1. Die WAP-Äquivalente für Desktop und Tabs

Soll die Funktionalität des Portals auf einem PDA oder einem Mobiltelefon zur Verfügung gestellt werden, ist erkennbar, dass das Portal auf diesen Geräten auf Grund der bekannten Einschränkungen wie geringer Displaygröße bzw. -auflösung und geringer Speicherkapazität nicht als Ganzes darstellbar ist.

Der Benutzer muss daher die Möglichkeit haben zu dem Track zu navigieren, der ihn interessiert. Dieser einzelne Track kann dann exklusiv am Gerät dargestellt werden.

Für die Navigation bietet sich die vorhandene hierarchische Struktur des Portals an. Dem Benutzer kann also zunächst die Liste der verfügbaren Tabs im Desktop präsentiert werden. Nach Auswahl eines Tabs gelangt er zur Liste aller sichtbaren

(und WAP-tauglichen) Tracks in diesem Tab. Von diesen Tracks kann dann wieder ein einzelner ausgewählt werden, der daraufhin ausgeführt und dargestellt wird.

Für die praktische Umsetzung dieses Ansatzes wurde der erste Schritt bereits getan: Die Funktion `wmlOutput()` (siehe Seite 70) ruft `hw_dc_checkAndExecuteShowMethod()` auf, um Dokumentklassen die Möglichkeit zu bieten, sich zu visualisieren. Intern wird dabei die Dokumentklassen-Methode `hw_gui_show()` aufgerufen, wie zuvor beschrieben wurde.

Die Methode `hw_gui_show()` der Dokumentklasse `HW_PortalDesktop` bietet sich daher als Weiche an – hier kann für WAP eine eigene Funktion geliefert werden, um den Code sauber von der Web-Version abzugrenzen. Die Methode ist im Folgenden ohne Fehlerbehandlung wiedergegeben:

```
function hw_gui_show(api, inargs) {
  var methodname = isWapClient() ? 'renderWirelessDesktopContent'
                                 : 'renderDesktopContent';

  var out = inargs.hw_this.classdefinition.methodsByName(methodname);
  inargs.method = out[0];
}
```

Die Darstellung des Desktops wird also für einen WAP-Client von der Dokumentklassen-Methode `renderWirelessDesktopContent()` übernommen. Diese Methode kann nun, ähnlich wie ein Collection Listing, eine Liste der Kinder des Desktops, also der Tabs des Portals, präsentieren. Dabei wird wieder die Funktion `buildWmlList()` verwendet, um die maximale Deckgröße des Clients nicht zu überschreiten.

Im Folgenden ist diese Methode ohne Fehlerbehandlung abgedruckt. Der Kürze wegen wird in diesem Beispiel auf die Bestimmung der Tabs und deren Reihenfolge verzichtet:

```
function renderWirelessDesktopContent(api, inargs) {
  var tabs = ...;
  var logo = '<img src="' + ... + '"/>';

  var wml_head = beginWml() + beginCard("main", "Welcome to...") +
    wmlCommonMenu() + '<p>' + logo + '<br/>';
  var wml_tail = '</p>' + endCard() + wmlInfoCard() + endWml();

  var remaining_size =
    wapMaxDeckSize() - estimatedBytecodeSize(wml_head) -
    estimatedBytecodeSize(wml_tail);
```

```
  function getLine(i) {
    var tab = tabs[i];
    return '<a href="' + getShortestObjectPath(tab) + '">' +
      tab.title(client.language) + '</a><br/>';
  }

  var start = getParametersFromUrl(request.url)['start'];
  var end = tabs.length - 1;
  wml_main = buildWmlList({start:start, end:end,
    size:remaining_size, getLine:getLine});

  write(wml_head, wml_main, wml_tail);
}
```

Es wird also eine Liste von Hyperlinks auf die einzelnen Tabs erstellt. Wenn der Benutzer einen dieser Links verfolgt, wird das zugehörige Objekt, eine Instanz von `HW_PortalTab`, vom Server angefordert. Um die oben beschriebene Navigation zum gewünschten Track zu realisieren, muss nun in analoger Weise eine Liste von Hyperlinks auf die einzelnen Tracks erstellt werden.

`HW_PortalTab` braucht also selbst wieder eine `hw_gui_show()`-Methode, die im Falle eines WAP-Clients eine andere Methode zurückliefert, die die Darstellung der Dokumentklasse übernimmt. Dies war für die Darstellung des Portals im Web-Browser nicht notwendig, da dabei die Tab-Instanzen nicht vom Client, sondern vom übergeordneten Desktop angefordert wurden.

Die Methode `hw_gui_show()` liefert in diesem Fall die Methode `renderWirelessTabContent()` zurück. Beide Methoden sind in ihrer Funktionsweise analog zu denen, die oben für die Dokumentklasse `HW_PortalDesktop` beschrieben wurden und brauchen an dieser Stelle nicht wiedergegeben werden.

10.1.1. Laufzeitoptimierung der Tab-Liste

Der bisher gezeigte Aufbau von Tab- und Tracklisten funktioniert zwar in der Praxis, birgt aber ein kleines Problem in sich. Dessen Ursache liegt darin, dass nicht jede Subklasse von `HW_PortalTrack` den drahtlosen Zugang unterstützen muss. Ohne auf die im nächsten Abschnitt 10.2 beschriebenen Details eingehen zu wollen, sei vorweggenommen, dass ein Track nur dann auf drahtlosem Wege verwendbar ist, wenn er die Methode `buildWirelessCards()` implementiert.

Dem Benutzer eines WAP-Gerätes sollten nur diejenigen Tracks angeboten werden, die den WAP-Zugang unterstützen. Tabs, die keine einzige WAP-fähige Track-Instanz beinhalten, sollten ebenfalls gar nicht erst in der Tab-Liste des Portals aufscheinen.

Jede Hyperwave-Dokumentklasse verfügt über eine Klassendefinition, die unter anderem die Signaturen aller Methoden der Klasse beinhaltet. Diese Klassendefinition kann zur Laufzeit vom API erfragt und analysiert werden. Die folgende Funktion zeigt, wie dieser Mechanismus verwendet werden kann, um einen Track auf das Vorhandensein der Methode `buildWirelessCards()` zu testen:

```
function isWirelessTrack(track) {
  var res = api.classDefinition(classname:track.HW_DC_InstanceOf);
  var cd = res.classdefinition;
  var methods = cd.methodsByName('buildWirelessCards');
  return methods.length > 0;
}
```

Damit können in die Liste der Tracks eines Tabs relativ einfach nur diejenigen aufgenommen werden, die dieses Kriterium erfüllen. Die Liste der Tabs des Desktops bleibt allerdings ein Problem, da die Tabs aus Performancegründen nicht alle ihre Tracks auf die beschriebene Weise testen können, nur um zu erfahren, ob mindestens ein WAP-fähiger Track darunter ist.

Als Lösung bietet sich an, diese Information in einem booleschen Objektattribut des Tabs abzulegen und dafür zu sorgen, dass sie bei Änderung aktualisiert wird. Dann braucht bei der Ausgabe der Tab-Liste nur dieses Attribut betrachtet werden, anstatt alle Tracks zu prüfen. Das Attribut besitzt in der Implementierung den Namen `has_wireless_tracks`.

Offensichtlich kann sich diese Information nur dann ändern, wenn Tracks in den Tab eingefügt oder aus ihm gelöscht werden. Die Dokumentklassen erlauben es, diese beiden Ereignisse zu überwachen, indem die beiden Methoden `onInsert()` und `onRemove()` überladen werden. Innerhalb von `onInsert()` wird `has_wireless_tracks` auf `true` gesetzt, wenn das einzufügende Objekt ein WAP-fähiger Track ist. Umgekehrt wird das Attribut in der Methode `onRemove()` auf `false` gesetzt, wenn das zu löschende Objekt der letzte WAP-fähige Track des Tabs war.

10.2. Die Darstellung eines Tracks

Wenn der Benutzer ein Element aus der Liste der Tracks eines Tabs auswählt, wird das zugehörige Objekt geladen. Dieses Objekt ist eine Instanz einer von `HW_PortalTrack` abgeleiteten Dokumentklasse. Damit der Track WML ausgeben kann und somit WAP-fähig wird, muss auch für ihn eine Ausgabemethode deklariert werden, indem `hw_gui_show()` in bekannter Weise angepasst wird.

Damit wäre die Erstellung WAP-fähiger Tracks grundsätzlich ermöglicht. Die Hersteller von Tracks für das Hyperwave eKnowledge Portal könnten zusätzlich zu `renderTrackContent()` für die Ausgabe von HTML eine zweite Dokumentklassen-Methode zur Ausgabe von WML implementieren und diese dem System mittels `hw_gui_show()` bekannt geben.

Im Web-Browser ist ein Track jedoch mehr als nur eine Dokumentklasse, die innerhalb einer HTML-Tabelle ausgegeben wird. Die Unterstützung des Frameworks geht darüber hinaus, und das sollte sie auch im Falle des WAP-Zugangs tun. Das Framework generiert im Web-Browser auch ein Menü für jeden Track, über das er unter anderem konfiguriert werden kann. Eine Erweiterung dieses Menüs durch den Track selbst ist ebenfalls möglich. Außerdem bietet das Framework Unterstützung für die Erstellung von Hilfetexten und die Einbindung eines About-Dialogfensters.

Eine vergleichbare Funktionalität lässt sich für WAP erreichen, indem die Basisklasse `HW_PortalTrack` selbst das WML-Deck generiert und lediglich die Erstellung einzelner Cards dieses Decks an eine konkrete Implementierung eines Tracks (eine Subklasse) vergeben wird. Die Basisklasse kann anschließend eigene Menüpunkte in die von der Subklasse generierten Cards integrieren und eigene Cards hinzufügen, um eine konsistente Benutzerschnittstelle zu gewährleisten.

Die in `HW_PortalTrack` implementierte Methode zur Ausgabe des gesamten Decks trägt den Namen `renderWirelessDeck()` und wird dem System wie beschrieben durch `hw_gui_show()` bekannt gegeben. Diese Methode ruft die von einer Subklasse zu implementierende Methode `buildWirelessCards()` auf, deren Aufgabe es ist, die Cards des Decks zu erzeugen. In `buildWirelessCards()` darf die Ausgabe nicht wie sonst durch Aufrufe von `write()` oder dergleichen erfolgen, sondern sie muss als String an die aufrufende Methode zurückgeliefert werden, um von dieser bearbeitet und um ein Menü erweitert werden zu können.[1]

[1] Dieser Umstand soll durch das Präfix „build“ an Stelle von „render“ ausgedrückt werden.

Bei Einhaltung dieser Konvention sind Tracks also genau dann WAP-fähig, wenn sie die Methode `buildWirelessCards()` implementieren, wie bereits im letzten Unterabschnitt 10.1.1 auf Seite 81 erwähnt wurde.

Im Folgenden wird `renderWirelessDeck()` mit ihren Hilfsfunktionen (wie gewohnt ohne Fehlerbehandlung) wiedergegeben und schrittweise erklärt; den Anfang bildet `renderWirelessDeck()` selbst:

```
function renderWirelessDeck(api, inargs) {
  var parameters = getParametersFromUrl(request.url);
  var methodname = parameters._method;

  if (methodname)
    callCustomMethod(inargs, methodname, parameters);
  else
    callBuildWirelessCards(inargs, parameters);
}
```

Wie man sieht, wird hier zunächst betrachtet, ob die URL (mit der die Track-Instanz durch den Client angefordert wurde) einen Parameter namens `_method` beinhaltet. Ist dies der Fall, wird die weitere Programmausführung an `callCustomMethod()` delegiert, ansonsten an `callBuildWirelessCards()`. Dieser einfache Mechanismus ermöglicht es, andere Dokumentklassen-Methoden zur Ausgabe des Deck-Inhalts verwenden zu können. Ein Track-Entwickler kann solche Methoden in das Menü des Tracks einbinden, wie unten gezeigt wird.

Die Funktion `callCustomMethod()` ist sehr einfach. Der `_method`-Parameter wird aus der Liste gelöscht, um ihn vor der auszuführenden Methode zu verbergen und `api.call()` wird verwendet, um die angegebene Methode aufzurufen:

```
function callCustomMethod(inargs, methodname, parameters) {
  delete parameters._method;

  var out = api.call({classname:inargs.hw_this.classdefinition.name,
    methodname:methodname, parameters:parameters});
}
```

Im anderen Fall wird die Hilfsfunktion `callBuildWirelessCards()` ausgeführt, deren Aufgabe es ist, die in der Subklasse implementierte Methode `buildWirelessCards()` aufzurufen. Hier wird zunächst das Grundgerüst des Decks festgelegt und seine Bytecode-Größe bestimmt. Die für die Cards verbleibende maximale Größe wird dann als zusätzlicher Parameter an `buildWirelessCards()` übergeben. Nach

der Ausführung von `api.call()` wird das Ergebnis, also die von der Track-Instanz erzeugten Cards, in das Deck-Grundgerüst integriert und ausgegeben:

```
function callBuildWirelessCards(inargs, parameters) {
  var object = inargs.hw_this.object;
  var goid = object.getGOid();
  var uri = getShortestObjectPath(object);

  var t_menu = createMenuForTrack(api, object, goid) +
    wmlCommonMenu();
  var t_head = beginWml();
  var t_tail = wmlInfoCard() + endWml();

  var used_size = estimatedBytecodeSize(t_head) +
    estimatedBytecodeSize(t_menu) + estimatedBytecodeSize(t_tail);
  parameters._size = wapMaxDeckSize() - used_size;

  var out = api.call({classname:inargs.hw_this.classdefinition.name,
    methodname:'buildWirelessCards', parameters:parameters});

  var t_cards = out.parameters._content;

  var index = t_cards.indexOf('</card>');
  write(t_head, t_cards.substring(0, index), t_menu,
    t_cards.substring(index), t_tail);
}
```

Nun kann das schon mehrfach angesprochene Menüsystem beschrieben werden. Vor dem Aufruf von `buildWirelessCards()` wird eine Hilfsfunktion namens `createMenuForTrack()` verwendet, um dieses Menü zu erstellen. Das Menü besteht aus einer Liste von `do`-Elementen der Sprache WML (vgl. Kapitel 2.2 auf Seite 19).[2]

Die Syntax von WML verlangt, dass das `do`-Element innerhalb eines `card`-Elementes stehen muss. Das macht einen kleinen Kunstgriff erforderlich, da die Cards von der Subklasse und die Menüs von der Basisklasse erstellt werden. Im oben angeführten Code wird daher mit ‚`t_cards.indexOf('</card>')`' nach dem Ende der ersten Card gesucht und das Menü an dieser Stelle eingefügt.

Im Folgenden ist noch die Hilfsfunktion `createMenuForTrack()` wiedergegeben, die die Menüeinträge zu erstellen hat:

[2] Wie diese Elemente dem Benutzer präsentiert werden, ist Sache des WML-Browsers; möglich sind Buttons ebenso wie Pull-Down-Menüs.

```
function createMenuForTrack(api, object, goid) {
  var hw_dc_methods = api.call({objectidentifier:goid,
    methodname:'hw_wap_methods'});

  var methods = hw_dc_methods.parameters.methods;

  var menu = '';

  for (var i in methods) {
    var method = methods[i];
    var caption = getCaption({caption:method.caption,
      name:method.name});
    menu += '<do type="options" label="' + caption + '"><go ' +
      'href="/' + goid + '?_method=' + method.name + '"/></do>';
  }

  return menu;
}
```

Zunächst wird hier die Dokumentklassen-Methode `hw_wap_methods()` aufgerufen. Aufgabe dieser Methode ist es, eine Liste von Methoden zurückzuliefern, die in das Menü des Tracks eingebunden werden sollen. Anschließend wird für jede dieser Methoden ein `do`-Element erzeugt, das an einen `go`-Task gebunden wird.

Die Ziel-URL des `go`-Tasks ist die Track-Instanz selbst, der Name der aufzurufenden Methode wird an die URL als Parameter `_method` angehängt. Wenn der Benutzer einen dieser Menüpunkte auswählt, wird der Track also erneut geladen. Wie oben gezeigt wurde, überprüft `renderWirelessDeck()` dann das Vorhandensein des URL-Parameters `_method` und ruft die angegebene Method auf, wenn das der Fall ist.

Damit hat der Track-Entwickler die Möglichkeit, auf einfache Weise zusätzliche Dokumentklassen-Methoden in das Menü des Tracks zu integrieren, die dann vom Benutzer aufgerufen werden können. Er braucht dazu lediglich die Methode `hw_wap_methods()` zu überladen und die Liste der einzubindenen Methoden zurückzugeben.

Es sei noch erwähnt, dass auf diese Weise eingebundene Methoden wieder die volle Kontrolle über das zu erzeugende Deck besitzen und nicht nur einzelne Cards erstellen dürfen, wie das bei `buildWirelessCards()` der Fall ist. Die Ausgabe des WML-Codes hat auch durch Aufruf von `write()` zu erfolgen und muss nicht als String zurückgegeben werden.

10.3. Ausgabe von Hilfetexten

Bei der Darstellung eines Tracks im Web-Browser bietet das Framework auch Unterstützung bei der Ausgabe von Hilfetexten in der bevorzugten Sprache des Benutzers. Im Allgemeinen können bei Hyperwave für die Ausgabe in verschiedenen Sprachen so genannte Language Resource Files mit der Dateiendung `.lrc` verwendet werden. Für jede Sprache, die unterstützt werden soll, muss dann eine solche Datei existieren, die in einfacher Weise Schlüssel auf Zeichenketten in der jeweiligen Sprache abbildet. Diese Language Resource Files können verwendet werden, nachdem sie mittels der JavaScript-Funktion `resdef()` eingebunden wurden. Die Verwendung erfolgt dann durch Aufruf der Funktion `lrc()`, die als Parameter neben dem Schlüssel auch eine Voreinstellung für den Rückgabewert erfordert. Dieser Vorgabewert wird zurückgegeben, wenn das Language Resource File den angeforderten Schlüssel nicht beinhaltet.

Das Hilfesystem der Tracks bei der Darstellung im Web-Browser ist in der Art realisiert, dass zwei bestimmte Schlüssel für den Hilfetext und die zugehörige Überschrift reserviert sind, nämlich `general` und `general_dlgCap` (für *dialog caption*). Wenn der Benutzer die Hilfe aktiviert, werden die beiden zugehörigen Zeichenketten aus der richtigen `lrc`-Datei ausgelesen und in einem eigenen Dialogfenster angezeigt.

Für die WAP-Darstellung eines Tracks ist die Ausgabe dieser Hilfetexte ebenfalls auf relative einfache Weise möglich. Zunächst muss die Hilfe in das Menü des Tracks eingebunden werden. Dies geschieht, indem `hw_wap_methods()` in der Basisklasse `HW_PortalTrack` so implementiert wird, dass sie eine Methode zur Ausgabe der Hilfe zurückliefert, wie im Folgenden gezeigt wird:[3]

```
function hw_wap_methods(api, inargs) {
  inargs.methods = new HW_DC_API_MethodArray();

  var cdef = inargs.hw_this.classdefinition;
  inargs.methods.append(cdef.methodsByName('wap_help')[0]);
}
```

Es wird also die Methode `wap_help()` in das Menü des Tracks integriert. Diese Methode muss dann das Language Resource File laden, die Werte der beiden

[3]Tracks, die eigene Methoden ins Menü integrieren wollen, können `hw_wap_methods()` überladen, sollten dabei aber die Implementierung der Basisklasse aufrufen und ihre eigenen Einträge an das Ende der aus dem Aufruf resultierenden Liste anhängen.

Schlüssel `general` und `general_dlgCap` ermitteln und das Ergebnis als WML-Deck ausgeben, wie der folgende Sourcecode zeigt:

```
function wap_help(api, inargs) {
  var class_name = inargs.hw_class.name;

  resdef('portal/tracks/' + class_name + '/langres/track.lrc',
    class_name);

  var help = lrc("general", class_name, "No help available.");
  var caption = lrc("general_dlgCap", class_name, "Help");

  write(beginWml(), beginCard('main', htmlToWml(caption)));
  write('<p>', htmlToWml(help), '</p>');
  write(endCard(), endWml());
}
```

Auf diese Weise werden die gleichen Hilfetexte wie bei der Darstellung im Web-Browser verwendet. Da dort aber HTML-Befehle erlaubt sind, müssen diese für den WML-Browser entfernt werden. Das erledigt die Funktion `htmlToWml()` auf die Weise, wie sie im Abschnitt 9.4 auf Seite 73 beschrieben wurde.

10.4. Ein Beispiel: `HW_PortalTrackPopMail`

Zum Abschluss dieses Kapitels und der gesamten Arbeit sollen die bisherigen Ausführungen zur Anpassung des Hyperwave eKnowledge Portals durch ein praktisches Beispiel für einen Track ergänzt werden. Damit soll neben der korrekten Implementierung von `buildWirelessCards()` auch gezeigt werden, wie eine andere Dokumentklassen-Methode zur Ausgabe verwendet werden kann, ohne sie in das Menü des Tracks zu integrieren.

Als Beispiel wurde ein Track gewählt, der es dem Benutzer ermöglicht, Nachrichten von einem POP3-E-Mail-Konto abzurufen. Dieser Track wurde unter dem Namen `HW_PortalTrackPopMail` für Hyperwave implementiert, wird hier jedoch zum besseren Verständnis in einer stark vereinfachten Version wiedergegeben. Wie in den bisherigen Code-Beispielen wird auch hier die Fehlerbehandlung außer Acht gelassen.

Der Track verwendet zur Abfrage der Nachrichten vom POP3-Server eine in JavaScript implementierte Hilfsklasse namens `PopClient`. Der Sourcecode dieser Klasse wird hier nicht abgedruckt, da er für das Verständnis der Funktionalität des Tracks

nur von sekundärem Interesse ist. Es sei jedoch erwähnt, dass die Klasse `PopClient` über die Methoden `open()`, `close()`, `getHeaders()` und `getMessage()` verfügt. Die Methoden `open()` und `close()` dienen zum Öffenen und Schließen der Verbindung zum POP3-Server; `open()` liefert außerdem die Anzahl der vorhandenen E-Mail-Nachrichten zurück. Mit `getHeaders()` können die Kopfzeilen einzelner Nachrichten abgefragt werden, und `getMessage()` dient zur Abfrage einer gesamten E-Mail.

Die Abfrage und Darstellung der E-Mail-Nachrichten gliedert sich in zwei Teile: Zunächst muss dem Benutzer eine Liste der Nachrichten präsentiert werden. Diese Liste sollte mit der aktuellsten Nachricht beginnen, da es mit dem mobilen Kommunikationsgerät mühsam sein kann, an das Ende einer langen Liste zu blättern. Nachdem der Benutzer eine Nachricht ausgewählt hat, muss ihr Inhalt in einem zweiten Schritt dargestellt werden. Bei der Darstellung der Nachricht sollte auch ein Hyperlink erstellt werden, der zurück auf die Liste an die Stelle der nächsten E-Mail verweist, um dem Benutzer die Navigation zu erleichtern.

Die hier abgedruckte Implementierung des Tracks verwendet für die Durchführung dieser beiden Teilaufgaben jeweils eine eigene Dokumentklassen-Methode. Die Darstellung der Nachrichtenliste wird von `buildWirelessCards()` übernommen (wodurch sich der Track in das Framework integriert), während die Ausgabe einer einzelnen Nachricht von einer eigenen Methode namens `showMail()` erledigt wird. Die beiden Methoden werden in den folgenden Unterabschnitten beschrieben.

10.4.1. Ausgabe der Nachrichtenliste durch `buildWirelessCards()`

Wenn der Benutzer eine Instanz des `HW_PortalTrackPopMail`-Tracks aktiviert, ruft das Framework die Dokumentklassen-Methode `buildWirelessCards()` auf, deren Aufgabe es ist, mindestens eine Card für das resultierende WML-Deck zu erstellen. Diese Card muss als String an das Framework zurückgegeben werden, indem sie der Variablen `inargs._content` zugewiesen wird.

Die vorliegende Implementierung der Methode stellt zunächst eine Verbindung zum POP3-Server her und ermittelt die Anzahl der verfügbaren Nachrichten. Dazu werden der Benutzername und das Passwort für das E-Mail-Konto sowie die Adresse des POP3-Servers benötigt. Diese Informationen werden als Attribute der Track-

Instanz gespeichert und müssen vom Benutzer bei der Instanzierung des Tracks im Web-Browser bekannt gegeben werden.[4]

Wenn keine Nachrichten vorhanden sind, wird eine entsprechende Meldung ausgegeben, anderenfalls wird die Hilfsfunktion `createInboxCard()` aufgerufen, um die Liste der Nachrichten auszugeben. Am Schluss wird die Verbindung zum POP3-Server geschlossen.

Die Nachrichtenliste kann länger sein, als es die maximale Bytecode-Größe des Clients gestattet. Um in der Liste blättern zu können, wird der Parameter `mailid` verwendet, der in der URL des Tracks angegeben werden kann und die Nachricht kennzeichnet, mit der die Listenausgabe begonnen wird. Die Methode `buildWirelessCards()` ist im Folgenden wiedergegeben:

```
function buildWirelessCards(api, inargs) {
  var username = api.get("username").value;
  var password = api.get("password").value;
  var pop3_server = api.get("pop3_server").value;

  var popclient = new PopClient(server:pop3_server, user:username,
    password:password, timeout:10);

  var title = inargs.hw_this.object.title(client.language);
  var size = inargs._size;

  var num_messages = popclient.open().messages;

  var start = inargs.mailid;
  if (start == undefined || start >= num_messages)
    start = num_messages - 1;

  inargs._content = start < 0
    ? wmlSimpleCard('main', title, "Your mailbox is empty")
    : createInboxCard(popclient, start, num_messages, size, title);

  popclient.close();
}
```

Die angesprochene Hilfsfunktion `createInboxCard()` übernimmt dann die Erstellung der Card. Um die maximale Deckgröße des WML-Browsers nicht zu überschreiten, wird `buildWmlList()` für die Ausgabe verwendet, nachdem die für die

[4] Da bei dieser einfachen Implementierung auch das Passwort unverschlüsselt gespeichert wird, empfiehlt es sich, die Track-Instanz selbst durch geeignete Rechtevergabe vor unbefugtem Zugriff zu schützen.

Liste verbleibende maximale Größe bestimmt wurde (vgl. Abschnitt 8.2 auf Seite 62). Dazu wird der Wert verwendet, den das Framework als maximale Größe für die Cards an `buildWirelessCards()` übergeben hat, und von diesem wird der anderweitig verwendete Speicherplatz abgezogen.

In der Liste werden Hyperlinks für jede einzelne Nachricht erstellt. Ziel dieser Links ist die Track-Instanz selbst, wobei aber der URL-Parameter `_method` auf den Wert `showMail` gesetzt wird. Als zweiter URL-Parameter wird `mailid` auf die Nummer der jeweiligen Nachricht gesetzt. Diese Vorgangsweise bewirkt, dass das Framework die angegebene Methode `showMail` anstelle von `buildWirelessCards()` aufruft, um das WML-Deck zu generieren, und ihr die übrigen URL-Parameter (in diesem Fall also die Nachrichtennummer) übergibt, wie im Folgenden gezeigt wird:

```
function createInboxCard(popclient, start, messages, size, title) {
  var t_head = beginCard("main", title) + "<p>" +
    (messages-start) + "-xxxxx of " + messages + " messages:<br/>";
  var t_tail = "</p>" + endCard();

  var remaining_size = size - (estimatedBytecodeSize(t_head) +
    estimatedBytecodeSize(t_tail));

  function getLine(mailid) {
    var popres = popclient.getHeaders(mailid);

    var from = popres.getAddresses(popres.getHeaderField('From'));
    from = from.length == 0 ? "unknown" : from[0].getShortAddress();

    var subject =
      decodeEncodedWords(popres.getHeaderField('Subject'));
    if (subject == undefined || subject.length == 0)
      subject = "[no subject]";
    if (subject.length > 25)
      subject = subject.substring(0, 22) + "...";
    subject = wmlEscape(subject);

    var url = getShortestObjectPath(request.object) +
      "?_method=showMail&amp;mailid=" + mailid;

    return "<tr><td><b>" + from + '</b></td><td><a href="' + url +
      '">' + subject + "</a></td></tr>";
  }

  var params = start:start, end:0, size:remaining_size,
    limit:5, getLine:getLine, parameter:'mailid', table:true;
```

```
  var t_main = buildWmlList(params);

  var end = messages - start + params.lines - 1;
  t_head = t_head.replace('xxxxx', end);

  return t_head + t_main + t_tail;
}
```

10.4.2. Ausgabe einer Nachricht mittels `showMail()`

Wie im letzen Unterabschnitt beschrieben wurde, wird eine eigene Dokumentklassen-Methode zur Ausgabe einer einzelnen E-Mail-Nachricht verwendet. Diese Vorgangsweise ist nicht zwingend erforderlich, sondern dient vielmehr zur Illustration der Möglichkeit an sich. Es hätte ebenso gut ein zusätzlicher URL-Parameter verwendet werden können, um die Ausgabe der Nachricht in `buildWirelessCards()` zu veranlassen. Bei der gewählten Variante muss die Methode `showMail()` im Unterschied zu `buildWirelessCards()` das gesamte Deck erstellen, nicht nur einzelne Cards.

Zunächst stellt die Methode die Verbindung zum POP3-Server her und lädt die im URL-Parameter `mailid` angegebene E-Mail herunter. Wenn es sich bei der E-Mail um eine mehrteilige Nachricht handelt, wird eine Fehlermeldung ausgegeben, dass solche Nachrichten nicht unterstützt werden.[5] Anderenfalls wird die Hilfsfunktion `createMailDeck()` aufgerufen, um die Nachricht anzuzeigen:

```
function showMail(api, inargs) {
  var mailid = Number(inargs.mailid);

  var username = api.get("username").value;
  var password = api.get("password").value;
  var pop3_server = api.get("pop3_server").value;

  var popclient = new PopClient(server:pop3_server, user:username,
    password:password, timeout:10);
  popclient.open();

  var popres = popclient.getMessage(mailid);

  popclient.close();

  if (popres.isMultipart()) {
```

[5] Diese Einschränkung dient zur Vereinfachung des Beispiels. Die tatsächlich im Einsatz befindliche Version des Tracks ist in der Lage, den ersten Textteil einer mehrteiligen Nachricht anzuzeigen.

```
    writeWmlErrorDeck("Error", "Can't handle multipart messages...");
    return;
  }

  createMailDeck(popres, mailid);
}
```

Die Funktion `createMailDeck()` erzeugt das Deck inklusive einer Card und bestimmt den für die Nachricht maximal verbleibenden Platz. Wenn die Nachricht diese Größe überschreitet, wird sie am Ende abgeschnitten.[6]

Um auf einfache Weise zur nächsten Nachricht gelangen zu können, erstellt die Methode einen Hyperlink auf die Liste der E-Mail-Nachrichten. Das erste Element dieser Liste ist dann die Nachricht, die unmittelbar auf die dargestellte folgt. Dazu wird ein `go`-Task über das `do`-Element in das Menü des WML-Browsers eingebunden. Als Nachrichtennummer wird `mailid` weniger Eins übergeben. Der Grund dafür ist, dass Nachrichten beim Eintreffen aufsteigend nummeriert werden, das heißt, die neueste Nachricht hat die höchste Nummer.

Die Hilfsfunktion `createMailDeck()` ist als Nächstes wiedergegeben:

```
function createMailDeck(popres, mailid) {
  var header = createHeader(popres);

  var uri = getShortestObjectPath(request.object);
  var wml_head = beginWml() + beginCard("main", "Content") +
    '<do type="options" label="Next message">' + '<go href="' + uri +
    "?mailid=" + (mailid-1) + '" method="get"/></do><p>' + header;

  var wml_tail = "</p>" + endCard() + endWml();

  var max_size =
    wapMaxDeckSize() - (estimatedBytecodeSize(wml_head) +
    estimatedBytecodeSize(wml_tail));

  var body = popres.getBody().replace(/\r\n/g, "\n");

  if (body.length > max_size)
    body = body.substring(0, max_size);

  body = wmlEscape(body).replace(/\n/g, "<br/>");
```

[6] Mit relativ wenig Aufwand kann das Blättern durch längere Nachrichten ermöglicht werden, indem die Hilfsfunktion `buildWmlList()` in bekannter Weise eingesetzt wird. Das Abschneiden dient ebenfalls der Vereinfachung des Beispiels.

```
  write(wml_head + body + wml_tail);
}
```

Abschließend werden noch zwei andere Hilfsfunktionen präsentiert, die in den obigen Funktionen verwendet werden. Die eine ist `createHeader()`, die dafür sorgt, dass bestimmte Header der E-Mail vor dem Inhalt der Nachricht angezeigt werden:

```
function createHeader(popres) {
  function headerLine(name, header) {
    if (!header)
      return "";
    return "<b>" + name + "</b>: " + wmlEscape(header) + "<br/>";
  }

  function addressLine(name, addresses) {
    if (addresses.length == 0)
      return "";

    var count = Math.min(addresses.length, 3);
    var result = "<b>" + name + "</b>: ";

    for (var number = 0; number < count; number ++) {
      var address = addresses[number];
      result += wmlEscape(address.getShortAddress());
      if (number < count - 1)
        result += ", ";
    }

    if (count < address.length)
      result += ", ...";

    return result + "<br/>";
  }

  var date = popres.getHeaderField("Date");
  var from = popres.getAddresses(popres.getHeaderField("From"));
  var to = popres.getAddresses(popres.getHeaderField("To"));
  var subject = decodeEncodedWords(popres.getHeaderField("Subject"));

  var header = headerLine("Date", date);
  header += addressLine("From", from);
  header += addressLine("To", to);
  header += headerLine("Subject", subject);
```

```
  return header + "<br/>";
}
```

Die andere Funktion ist `wmlEscape()`. Sie sorgt dafür, dass bestimmte Zeichen wie etwa „<“ oder „>“, die möglicherweise Teil einer E-Mail sein können, durch entsprechende WML-Befehle ersetzt werden, um korrekt dargestellt zu werden:

```
function wmlEscape(string) {
  return string.replace(/&/g, "&amp;").replace(/</g, "&lt;")
    .replace(/>/g, "&gt;") .replace(/\"/g, "&quot;")
    .replace(/\'/g, "&apos;");
}
```

Der gezeigte E-Mail-Track funktioniert in der Praxis natürlich auch in einem Web-Browser. Der für die HTML-Ausgabe zuständige Code ist jedoch für diese Arbeit nebensächlich und wurde daher nicht wiedergegeben.

Es wurde jedoch gezeigt, dass ein Track mit Hilfe der WAP-Komponente und der vorgenommenen Anpassungen des Portals relativ schnell und einfach dahingehend erweitert werden kann, dass er auch auf drahtlosem Wege verwendbar ist und in ähnlicher Weise funktioniert wie im Web-Browser.

Anhang

Abkürzungsverzeichnis

Im Rahmen dieser Arbeit werden folgende Abkürzungen mit den angegebenen Bedeutungen verwendet:

API	Application Programming Interface
CGI	Common Gateway Interface
CPU	Central Processing Unit
DTD	Document Type Definition
DTMF	Dual Tone Multiple Frequency
EKP	(Hyperwave) eKnowledge Portal
EKS	(Hyperwave) eKnowledge Suite
ELS	(Hyperwave) eLearning Suite
GIF	Graphics Interchange Format
GPRS	General Packet Radio Service
GSM	Global System of Mobile Communication
HTML	Hypertext Markup Language
HTTP	Hypertext Transfer Protocol
IEEE	Institute of Electrical and Electronics Engineers
ISO	International Standards Organization
JPEG	Joint Photographic Experts Group
MFV	Mehrfrequenz-Wahlverfahren
MIME	Multipurpose Internet Mail Extension
OSI	Open Systems Interconnection
PCS	Personal Communication System
PDA	Personal Digital Assistant
PDC	Personal Digital Communications
PDF	Portable Document Format
PNG	Portable Network Graphics
RAM	Random Access Memory
RFC	Request For Comments

SMS	Short Message Service
SSL	Secure Sockets Layer
TCP	Transmission Control Protocol
TDMA	Time Division Multiple Access
TLS	Transport Layer Security
UDP	User Datagram Protocol
URI	Universal/Uniform Resource Identifier
URL	Uniform Resource Locator
USSD	(GSM) Unstructured Supplementary Service Data
WAE	Wireless Application Environment
WAP	Wireless Application Protocol
WBMP	Wireless Bitmap
WDP	Wireless Application Protocol
WML	Wireless Markup Language
WSP	Wireless Session Protocol
WTA	Wireless Telephony Application
WTAI	Wireless Telephony Application Interface
WTLS	Wireless Transport Layer Security
WTP	Wireless Transaction Protocol
WWW	World Wide Web
XML	Extensible Markup Language

Literaturverzeichnis

[1] *Information Technology – Open Systems Interconnection – Basic Reference Model: The Basic Model*. 1994. – ISO/IEC 7498-1

[2] WAP Forum, Ltd.: *WAP-100, Wireless Application Protocol Architecture Specification*. April 1998. – URL: http://www.wapforum.org/

[3] Fielding, R. [u. a.]: *Hypertext Transfer Protocol – HTTP/1.1*. Juni 1999. – URL: http://www.ietf.org/rfc/rfc2616.txt

[4] WAP Forum, Ltd.: *WAP-162, Wireless Session Protocol Specification*. November 1999. – URL: http://www.wapforum.org/

[5] WAP Forum, Ltd.: *WAP-133, Wireless Transaction Protocol Specification*. Juni 1999. – URL: http://www.wapforum.org/

[6] Dierks, T. ; Allen, C.: *The TLS Protocol*. Januar 1999. – URL: http://www.ietf.org/rfc/rfc2246.txt

[7] WAP Forum, Ltd.: *WAP-163, Wireless Transport Layer Security Specification*. November 1999. – URL: http://www.wapforum.org/

[8] WAP Forum, Ltd.: *WAP-158, Wireless Datagram Protocol Specification*. November 1999. – URL: http://www.wapforum.org/

[9] WAP Forum, Ltd.: *WAP-152, Wireless Application Environment Overview*. November 1999. – URL: http://www.wapforum.org/

[10] Bray, T. [u. a.]: *Extensible Markup Language (XML), W3C Proposed Recommendation*. Februar 1998. – URL: http://www.w3.org/TR/REC-xml

[11] WAP Forum, Ltd.: *WAP-155, Wireless Markup Language Specification*. November 1999. – URL: http://www.wapforum.org/

[12] Berners-Lee, T. [u. a.]: *Uniform Resource Identifiers (URI): Generic Syntax*. August 1998. – URL: http://www.ietf.org/rfc/rfc2396.txt

[13] WAP Forum, Ltd.: *WAP-153, Wireless Application Environment Specification*. November 1999. – URL: http://www.wapforum.org/

[14] Aho, Alfred V. [u. a.]: *Compilerbau*. 1999. – ISBN: 3486252941 (Teil 1) und 3486252666 (Teil 2)

[15] WAP Forum, Ltd.: *WAP-156, WMLScript Language Specification*. November 1999. – URL: http://www.wapforum.org/

[16] WAP Forum, Ltd.: *WAP-157, WMLScript Standard Libraries Specification*. November 1999. – URL: http://www.wapforum.org/

[17] WAP Forum, Ltd.: *WAP-169, Wireless Telephony Application Specification*. November 1999. – URL: http://www.wapforum.org/

[18] WAP Forum, Ltd.: *WAP-170, Wireless Telephony Application Interface Specification*. November 1999. – URL: http://www.wapforum.org/

[19] WAP Forum, Ltd.: *WAP-174, User Agent Profiling Specification*. November 1999. – URL: http://www.wapforum.org/