Diplomarbeit

im Studiengang Medieninformatik

an der Hochschule der Medien Stuttgart

Anbindung mobiler Endgeräte mittels standardisierter Web Services auf Basis von Java 2 Micro Edition

Betreuung:

Prof. Dr.-Ing. Johannes Maucher

Dr. Marcus Iwanowski

Vorgelegt am 7.8.2006 von:

Jörg Richter

Ehrenwörtliche Erklärung

Hiermit erkläre ich, dass die vorliegende Arbeit selbstständig und ohne fremde Hilfe angefertigt wurde. Es wurden nur die in der Arbeit ausdrücklich benannten Quellen und Hilfsmittel benutzt. Wörtlich oder sinngemäß übernommenes Gedankengut habe ich als solches kenntlich gemacht.

Stuttgart, 7.8.2006

Jörg Richter

Abstract

Kurzfassung

Das Ziel dieser Diplomarbeit ist die Realisierung eines Clients für mobile Endgeräte auf Basis von J2ME[1] unter Verwendung von MIDP 2[2]. Mit Hilfe dieses Clients sollen Anbindungen an Standard Web Services auf Basis von SOAP möglich sein. Die Client-Anwendung stellt Formulare zur Ein- und Ausgabe von Daten bereit. Diese Formulare werden zur Laufzeit anhand der dem Dienst zu Grunde liegenden WSDL erzeugt und die erfassten Daten werden bis zum Absenden dieser an den Server lokal auf dem Endgerät gespeichert.

Schlagworte

- Web Services
- J2ME
- mobile Endgeräte
- WSDL
- XML

1 J2ME siehe Kapitel 3.5.2
2 MIDP siehe Kapitel 3.5.2

Inhaltsverzeichnis

Abbildungsverzeichnis

Ausarbeitung

1 Einleitung

1.1 Problemstellung und Zielsetzung

Heute sind mobile Endgeräte aus dem täglichen Leben kaum mehr wegzudenken. Sowohl im privaten als auch im geschäftlichen Umfeld werden diese als Kommunikationsplattform ebenso genutzt wie für die Verwaltung von Terminen. Selbst E-Mail-Dienste werden, derzeit noch vorrangig im geschäftlichen Umfeld, immer weiter mobil und mit Push-Technologien[3] zum ständigen Begleiter.

Die Dienste Telefonie, SMS und mobile E-Mail sind jedoch nur einige der Einsatzmöglichkeiten. Immer mehr Unternehmen wünschen sich beispielsweise Produktionsdaten in Echtzeit auf ihrem mobilen Endgerät einsehen zu können oder bei Problemen (voll automatisiert) benachrichtigt zu werden. Es ist eine deutliche Steigerung des Interesses an diesen noch recht neuen Technologien festzustellen, das in den nächsten Jahren noch weiter vorangetrieben werden wird.

Im Umfeld der mobilen Endgeräte ist vor allem im Bereich der Usability (Handhabbarkeit) von Diensten noch einige Arbeit zu leisten, zumal Erfahrungen aus anderen Bereichen wie Desktop-PCs portiert werden müssen. Probleme stellen vor allem die Geräte mit ihren Ein-und Ausgabemöglichkeiten sowie die stark begrenzten Systemressourcen dar.

Um auf beliebige Daten eines Unternehmens zugreifen zu können werden Schnittstellen benötigt, die entsprechende Ressourcen aus den verschiedensten Systemen zur Verfügung stellen. Eine der hierbei eingesetzten Technologien stellen Web Services[4] dar. Da sie unabhängig von den verwendeten Systemen (Plattformen) sind eignen sich Web Services bestens als Schnittstelle zwischen diesen Systemen, zumal Web Services bereits von vielen Anbietern als Schnittstellen zur Verfügung gestellt werden.

---

3 Mittels **Push-Technologie** werden Daten, wie z.B. E-Mails zu dem mobilen Endgerät gesendet und müssen nicht vom mobilen Gerät in regelmäßigen Abständen abgefragt werden. Push-E-Mail wurde vor allem durch die Firma RIM (Research In Motion) mit ihren Blackberry-Geräten bekannt.

4 Auf **Web Services** wird in Kapitel 3.4.1-WSDL auf Seite 28 näher eingegangen.

Das Ziel dieser Diplomarbeit ist es beliebige Web Services auf einer möglichst großen Zahl mobiler Endgeräte nutzbar zu machen, ohne für jeden dieser Web Services eine neue Applikation erstellen zu müssen.

Hierfür ist es zunächst notwendig mit dem mobilen Endgerät eine Verbindung zum Anbieter eines Web Service aufbauen zu können. Diese Verbindung wird unter Einbindung des Internetprotokolls TCP/IP realisiert, das das GSM-Mobilfunknetz in Form von GPRS unterstützt.

Für beliebige, zum Zeitpunkt der Erstellung (Implementierung) der Anwendung noch nicht bekannte, Web Services sollen Ein- und Ausgabeformulare anhand ihrer Beschreibungsdatei (WSDL) erzeugt werden.

Daten wie die Beschreibungsdateien und ausgefüllte, noch nicht gesendete Eingaben sollen beim Verlassen der Anwendung dauerhaft (persistent) für eine erneute Verwendung gespeichert werden.

1.2 Aufbau der Diplomarbeit

Kapitel 2 (Analyse): In diesem Kapitel werden die Rahmenbedingungen für das weitere Vorgehen herausgearbeitet. Es bietet eine Übersicht der in Frage kommenden Typen von mobilen Endgeräten und den daraus entstehenden Möglichkeiten.

Kapitel 3 (Technologien): In diesem Kapitel werden die zur Verwendung kommenden Technologien vorgestellt. Neben den Grundprinzipien von Web Services wird auf die Restriktionen der mobilen Endgeräte in Verbindung mit J2ME eingegangen.

Kapitel 4 (Realisierung der Aufgabenstellung): Nachdem die Rahmenbedingungen festgelegt sind wird in diesem Kapitel darauf eingegangen, wie der Prototyp realisiert wird.

2 Analyse

2.1 Was bietet der Markt derzeit an Anwendungen?

Derzeit gibt es für die Kommunikation zwischen mobilen Endgeräten und Diensten (Web Services) meist nur spezielle, für einen Verwendungszweck angepasste Lösungen. Hierbei werden zum Beispiel E-Mail- oder Kalender-Daten übermittelt, Aktienkurse abgefragt oder News bezogen. Jedoch auch Anwendungen im Business-Bereich wie die Abfrage von Produktionsdaten oder die mobile Erfassung von Aufträgen und Arbeitszeiten sind bereits verfügbar.

Frameworks wie beispielsweise „mobile Objects“[5] der Firma *Extended Systems* können zum Austausch von Daten in J2ME Anwendungen genutzt werden. Hierfür muss jedoch eine entsprechende Server-Infrastruktur vorhanden sein, die vor allem für kleine Lösungen zu aufwendig ist oder mit zu hohen laufenden Kosten einhergeht.

Es ist jedoch bei den meisten auf dem Markt verfügbaren Anwendungen festzustellen, dass sie zum einen nur für einen Anwendungsfall erstellt wurden und zum anderen an bestimmte Geräte gebunden sind. (mehr zu dieser Problemstellung im Kapitel 2.2)

2.2 Auswahl der mobilen Endgeräte

Der Markt bietet eine Vielzahl unterschiedlichster mobiler Endgeräte die möglicherweise Verwendung finden könnten. Es folgt eine Klassifizierung nach Leistungsfähigkeit in aufsteigender Reihenfolge, auf der die Entscheidungsfindung mit basiert. Die Unterscheidungsmerkmale bei mobilen Endgeräten mit Kommunikationsmöglichkeiten via GSM, GPRS oder UMTS fließen immer mehr ineinander und wachsen stetig weiter zusammen. Es kann nur eine Momentaufnahme dargestellt werden, die bereits bestehende Geräte mit aufnimmt. Einzelne Geräte werden nicht betrachtet, da diese nach der Fertigstellung dieser Diplomarbeit schon wieder überholt sein dürften.

5 „mobile Objects“ stellt ein Framework dar, mit dessen Hilfe über eine „OneBridge Mobile Groupware“-Lösung der Firma Extended Systems Daten zwischen mobilen Endgeräten und einer Server-Komponente ausgetauscht werden können.

Mobiltelefon

Die klassischen Mobiltelefone, umgangssprachlich als „Handy“ bezeichnet, sind im Laufe der Zeit zu wahren Alleskönnern herangewachsen. Diese Geräte zeichnen sich vor allem durch ihr meist proprietäres[6] Betriebssystem aus. Alle Geräte dieser Klasse unterstützen mindestens Telefonie über das GSM-Netz.

Es gibt in der Familie der Mobiltelefone auch heute noch die einfachen, günstigen Geräte, die bereits ohne Vertragsbindung und Sponsoring durch Provider sehr günstig zu erwerben sind. Diese einfachen Mobiltelefone bieten meist nur das nötigste. So sind neben der Telefonfunktion mit Telefonbuch meist kleine Programme wie Wecker, Rechner oder einfache Spiele vorhanden. Das Display hat eine geringe Auflösung und bietet oft nur monochrome (einfarbige) Darstellung. Anwendungen können oft nicht installiert werden. Eine Datenverbindung mittels GPRS ist nicht immer vorhanden.

Die gehobeneren Geräte dieser Klasse bieten eine Vielzahl von Erweiterungen wie WAP und/oder HTML Browser, Adressbuch, Kalender, Mail, Spiele, Digital-Kamera und die Möglichkeit Anwendungen (meist J2ME - Midlets[7]) zu installieren. Zur Datenverbindung über das Funknetz steht GPRS zur Verfügung. Meist bieten die Geräte eine Infrarot-Schnittstelle. Seltener, aber dennoch weit verbreitet sind Bluetooth-Schnittstellen. Die Displays bieten eine Auflösung von über 100x100 Pixeln und sind für die Farbdarstellung geeignet.

Smartphones, MDA, Palm, Blackberry, ...

Es wird zunehmend schwieriger die Geräteklassen klar zu differenzieren. Mit den „Smartphones“ verschmolzen die Grenzen zwischen PDA (Personal Digital Assistent) und Mobiltelefon. Smartphones sind Mobiltelefone, die um zusätzliche Funktionen wie erweiterte Kalender, E-Mail-Clients und Adressbuchfunktionen erweitert sind. Sie besitzen zudem oftmals Touchscreens zur Navigation über die Bedienoberfläche und Eingabe von Daten. Smartphones sind somit eine Mischung aus PDA und Mobiltelefon. Je nach Hersteller finden sich bei diesen Geräte unterschiedliche Betriebssysteme. Der Hersteller Nokia setzt beispielsweise das Betriebssystem Symbian-OS ein.

Symbian OS ist ein für die Verwendung im Umfeld von Mobiltelefonen optimiertes

6 proprietär bezeichnet Individuelle Lösungen, die keinem allgemeinen Standard entsprechen.

7 J2ME-Anwendungen werden als sogenannte Midlets auf das Endgerät geladen (siehe Kapitel 3.5.2 Java 2 Micro Edition (J2ME) Seite 34)

und entwickeltes Betriebssystem. Es unterstützt eine Vielzahl von Hardware und kann durch die Hersteller für die jeweiligen Geräte angepasst werden. Symbian OS unterstützt neben Anwendungen, die in C++ entwickelt werden auch J2ME -Anwendungen. Die Unterstützung von J2ME in Symbian-OS geht auf die weite Verbreitung von Java-fähigen Endgeräten und die große Entwicklergemeinde zurück. Die Java-Laufzeitumgebung von Symbian OS wird ständig auf dem aktuellen und vom Markt geforderten Stand gehalten. Durch die Optimierung der Java-Laufzeitumgebung in Symbian OS v8.0 und die gesteigerten Taktraten der eingesetzten Hardware war es im Jahr 2004 möglich die Performance um etwa den Faktor 40 gegenüber der ersten Implementierung in Symbian OS v5 zu erhöhen. Die Zukunft soll hier noch weitere Performance-Steigerungen bringen.[SYMOS]

Andere Geräte wie beispielsweise die durch die Telekom vertriebenen MDA's verwenden vorwiegend „Windows Mobile“ der Firma Microsoft als Betriebssystem. Diese Windows-Version ging aus Windows CE, einer Windows-Variante für eingebettete-Systeme (Systemen mit geringen Ressourcen) hervor. Windows Mobile ist auf die Leistungsfähigkeit dieser Umgebung angepasst und unterstützt zahlreiche Varianten von Hardware. Anwendungen für Windows Mobile können als „CE .NET“ Implementierungen erstellt werden. Eine gute Java-Laufzeitumgebung ist für dieses System nicht zu finden und wird von Microsoft derzeit auch nicht gefördert. Auffällig bei Windows Mobile ist, dass im Gegensatz zu Symbian-OS das Betriebssystem in den Geräten durch das Erscheinungsbild der Bedienoberfläche klar zu erkennen ist (es lässt sich eine gewisse Ähnlichkeit mit anderen Windows Betriebssystemen erkennen).

Die Firmen Palm und RIM (Research In Motion) verwenden bei ihren Produkten eigene Betriebssysteme, wobei Palm mit dem Betriebssystem Palm-OS arbeitet und RIM ein proprietäres System einsetzt. Beide Systeme bieten eine Java-Laufzeitumgebung.

Die Firma RIM konnte große Marktanteile durch die in ihrer Blackberry-Produktfamilie verwendete E-Mail-Push-Funktion für sich gewinnen. Durch diese Push-Funktion war es erstmals möglich E-Mails nicht durch periodisches Abfragen nach neuen Nachrichten sondern durch eine Benachrichtigung seitens des Servers zu empfangen. Hierdurch kann unnötiger Datenverkehr verhindert und gleichzeitig die Aktualität erhöht werden.

Aufgrund der großen Anzahl von J2ME unterstützenden Endgeräten wird dieses als Basis für diese Diplomarbeit Verwendung finden. Es ist hierdurch möglich eine sehr große Anzahl auf dem Markt bereits vorhandener Endgeräte anzusprechen und wird auch in näherer Zukunft Unterstützung finden.

3 Technologien

In diesem Kapitel werden die für diese Diplomarbeit verwendeten Technologien vorgestellt.

Abbildung 1 stellt einen Web Service Stapel dar. Sie zeigt welche Protokolle und Technologien bei der Nutzung eines Web Service Verwendung finden. In der den Web Service beschreibenden WSDL sind die zur Nachrichtenübermittlung genutzten Protokolle ebenso spezifiziert wie der zur Übermittlung verwendete Übertragungskanal.

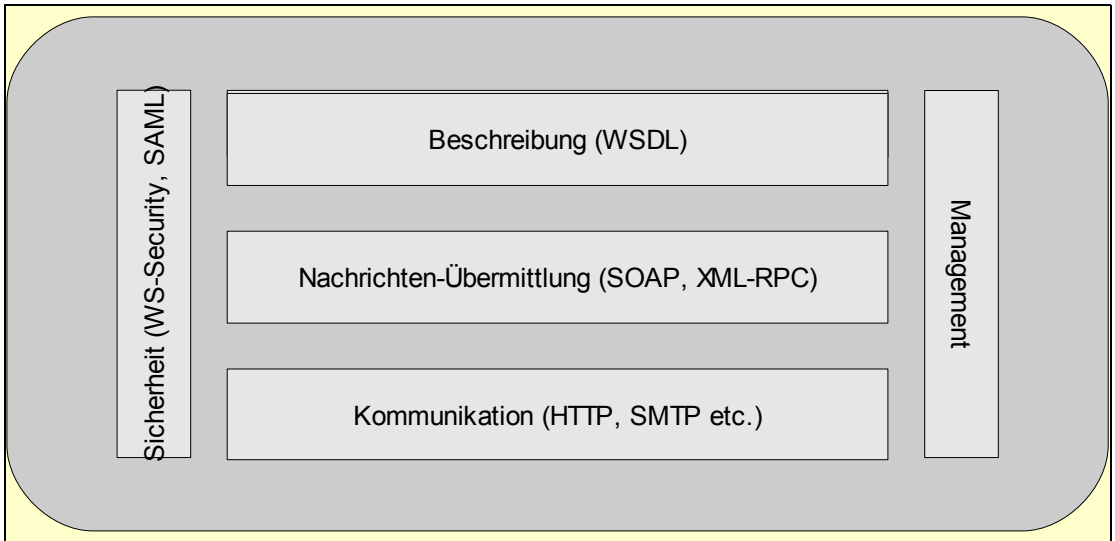


Abbildung 1: Web Service Stapel [WSS04]

Um diesen Web Service Stapel besser verstehen zu können werden zunächst einige Technologien näher beschrieben.

3.1 Markup Language-Familie

In den folgenden Unterkapiteln werden die Grundprinzipien der bei Web Services verwendeten Auszeichnungssprachen dargestellt. Die Basis für diese Sprachen stellt die Metasprache zu Auszeichnungssprachen - SGML dar.

3.1.1 Standard Generalized Markup Language (SGML)

SGML entwickelte sich aus proprietären Auszeichnungssprachen, die beispielsweise von Druckereien dazu genutzt wurden Dokumente mit Anmerkungen vom Autor für den Drucker zu versehen. Computer konnten nun dafür genutzt werden diese Doku-

mente zu lesen und zu drucken, ohne die Anmerkungen, die nicht Bestandteil des Textes waren, mit auf dem Ausdruck erscheinen zu lassen.

Die Meta-Auszeichnungssprache SGML entstand im Jahr 1985 und wurde als Inhaltsstruktur für Dokumente in der Norm ISO 8879 als Standard festgelegt.

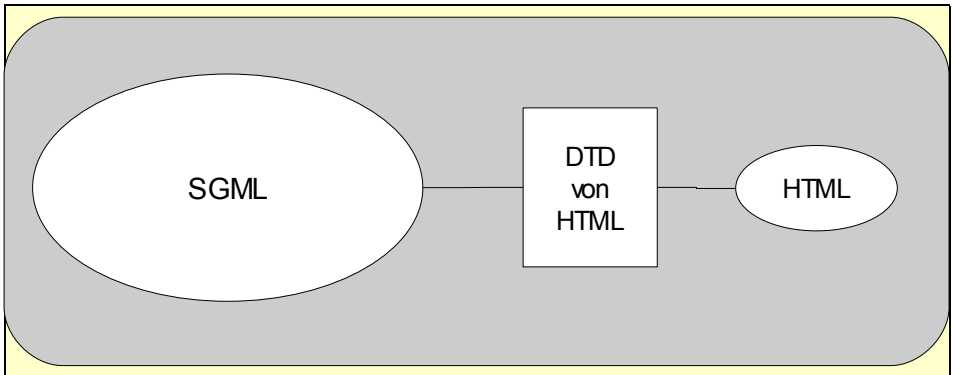


Abbildung 2: HTML stammt von SGML ab [RRZN01]

Dieser Standard wurde unter anderem als Grundlage für HTML verwendet. Bei HTML handelt es sich um einen in einer DTD[8] festgelegten Satz von Elementen mit dem Dokumente logisch ausgezeichnet werden können.

3.1.2 XML

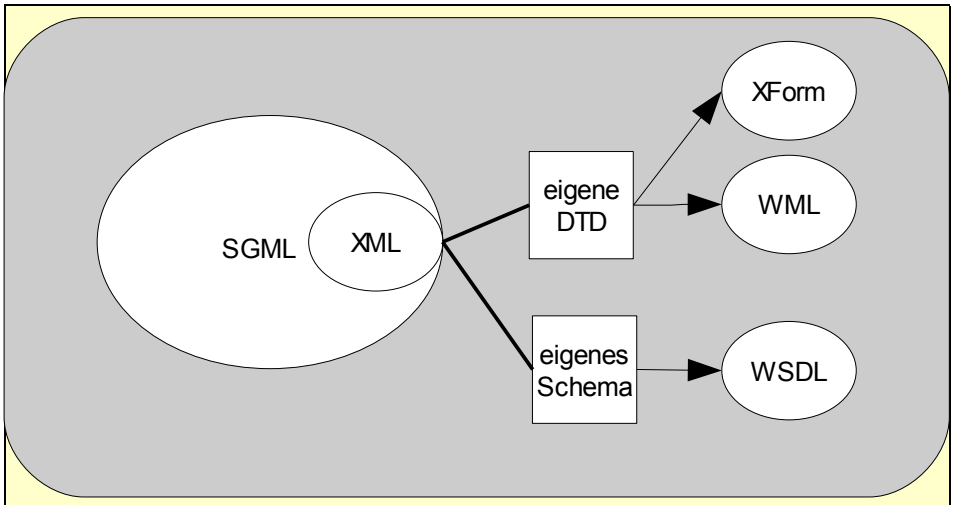


Abbildung 3: XML stellt eine Teilmenge von SGML dar.

XML (Extensible Markup Language) stellt wie in Abbildung 3 dargestellt eine Teilmenge von SGML dar. XML ist daher wie SGML sehr gut erweiterbar („extensible“). Es gibt keine feste Definition die Elemente betreffend, lediglich die Definition zum Aufbau der Beschreibungsstruktur ist vorgegeben. Eine solche

8 Die DTD stellt eine Definition zum Aufbau eines XML-Dokuments dar.(siehe Kapitel 3.1.3)

Beschreibungsstruktur kann als DTD oder Schema ausgeführt sein (später mehr dazu).

XML dient zum logischen Strukturieren von Dokumenten. Ein XML-Dokument stellt inhaltlich eine Baumstruktur dar, die für jedes Element öffnende und schließende „Tags“ (Kennzeichner) besitzt. Zusätzlich können zu den Elementen noch Attribute kommen, die dann einem bestimmten Element zugehörig sind.

Es folgt ein kleines Beispiel eines XML-Dokumentes an dem der strukturierte Aufbau zu erkennen ist:

```
1|<Dokument>
2|        <Titel Besitzer=“mein Name“>Adressen</Titel>
3|        <Eintrag>
4|                <Name>Müller</Name>
5|                <Vorname>Klaus</Vorname>
6|                <Anschrift Strasse=“Umweg“ Hausnummer=“7“ PLZ=“0815“ Ort=“Schönstadt“> </Anschrift>
7|                <EMail/>
8|        </Eintrag>
9|        <Eintrag>
10|               <Name>Hase</Name>
11|               <Vorname>Wolfgang</Vorname>
12|               <Anschrift Strasse=“Feldrand“ Hausnummer=“1“ PLZ=“1111“ Ort=“Kleindorf“> </Anschrift>
13|               <EMail>Hase@T-Hoppelbein.karotte</EMail>
14|       </Eintrag>
15|       <!-- hier steht ein Kommentar -->
16|</Dokument>
```

Abbildung 4: XML-Beispiel

Gut zu erkennen ist die Struktur mit den öffnenden <Tag-Name> und schließenden Tags </Tag-Name>. Für jedes öffnende Tag muss es auch ein schließendes Tag geben um die Struktur gültig zu halten. Jedoch sollte dem/der aufmerksamen Leser/in aufgefallen sein, dass in Zeile 7 scheinbar nur ein Tag verwendet wird. Bei dem Tag in Zeile 7 handelt es sich um eine spezielle Kurzschreibweise für Elemente, deren Inhalt leer ist. Es wird der Schrägstrich hierbei hinter den Tag-Namen gestellt (<Tag/>).

Ein besonderes Element stellt der in Zeile 15 zu sehende Kommentar dar. Ein Kommentar wird mit der Zeichenkette „<!--“ begonnen und endet mit der Zeichenkette „-->“. Kommentare werden für gewöhnlich von der XML-verarbeitenden Software nicht berücksichtigt.

Der große Vorteil dieser XML-Schreibweise ist die Lesbarkeit für Mensch und Maschine. Jeder sollte verstehen können worum es in dem Beispiel geht, nämlich um eine Auflistung von Adressen, bestehend aus Namen, Anschrift und E-Mail-Adresse. Mit einem Computer sind diese Informationen ebenso gut zu lesen, da alle Informationen ihre eigene Bezeichnung haben. Da es sich hierbei um eine reine Textdatei handelt ist sowohl die Bearbeitung bereits mit einfachsten Mitteln (einfacher Texteditor) als auch die System-/Plattformunabhängigkeit gewährleistet.

Am Beginn eines XML-Dokuments muss der so genannte Prolog stehen, der im obigen Beispiel nicht mit angezeigt wurde. Der Prolog gibt Auskunft über die Art des Dokuments und sollte folgende Form haben:
<?xml version="1.0" encoding="ISO-8859-1"?>
Es wird zunächst die XML-Version angegeben, gefolgt vom „encoding" also der Zeichenkodierung des Dokuments.

Die Struktur kann aufgrund des klar definierten Aufbaus des Dokuments von entsprechenden Editoren auf sogenannte Wohlgeformtheit überprüft werden. Eine Prüfung auf Wohlgeformtheit kann unabhängig von den im Dokument verwendeten Elementen und Attributen durchgeführt werden, da hierbei folgende Punkte überprüft werden:

- ein Prolog ist vorhanden,
- es gibt mindestens ein Element,
- es gibt nur ein sogenanntes „ROOT"-Element als Wurzel der Baumstruktur
- zu jedem öffnenden Tag ist auch ein schließendes Tag vorhanden,
- Tags „überlappen" sich nicht (also es kommt keine Struktur <a> im Dokument vor),
- Attributwerte stehen in einfachen (') oder doppelten (") Hochkommas
- Attributnamen sind pro Element nur einmal vorhanden.

Überprüfungen auf Ebene der Elemente und Attribute werden durch Verwendung einer DTD oder eines XML-Schemas möglich (siehe DTD).

Für die Verarbeitung durch Computer werden XML-Dokumente zunächst von einem so genannten Parser verarbeitet. Bei diesen Parsern gibt es zwei grundlegende Ansätze.

„DOM-Parser" erstellen anhand der Daten des XML-Dokuments einen DOM (Docu-

mernt Object Model) - Baum. In der Programmiersprache Java liegt beispielsweise nach dem Parsen mittels DOM-Parser ein Objekt vor, über das es möglich ist Kind-Knoten, also Unterelemente und dessen Attribute als Objekte abzufragen und zu bearbeiten.

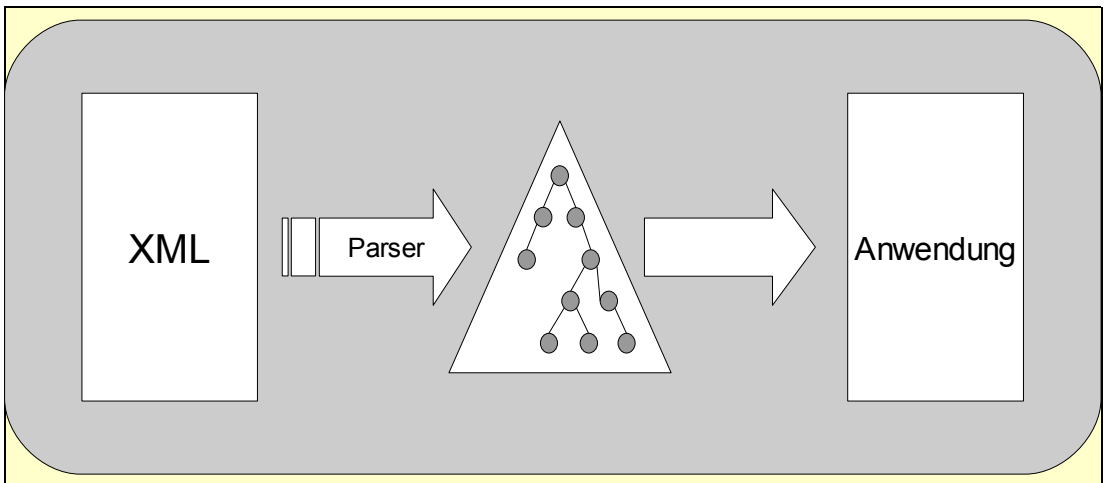


Abbildung 5: DOM-Parser

„SAX-Parser“ verfolgen einen etwas anderes Ansatz. So werden bei SAX (Simple API for XML) -Parsern keine Objekte mit all ihren Elementen erzeugt wie bei DOM-Parsern, sondern das Dokument durchlaufen (geparst) und anhand unterschiedlicher Ereignisse werden Aktionen ausgelöst. Solche Ereignisse entstehen durch Finden der folgenden Teil-Elemente: Anfang (Start) eines Elements, Inhalt eines Elements, Ende eines Elements, Kommentare, Entity-Definitionen (Beschreibungen von Sonderzeichen), Warnungen und Fehler.

Beide, sowohl DOM- als auch SAX-Parser sind in den meisten (wenn nicht sogar allen) Programmiersprachen bereits implementiert, was der programmiersprachenunabhängigen Verteilung von XML-Dokumenten sehr zugute kommt.

3.1.3 DTD

Die DTD (Dokumenttyp Definition) dient als Beschreibung für ein XML-Dokument. Zur Veranschaulichung das Beispiel HTML. HTML hat eine festgeschriebene Anzahl von Elementen, die in einer bestimmten Reihenfolge angegeben werden müssen und bestimmte Attribute haben dürfen bzw. haben müssen. Sozusagen einen Bauplan. Was wie und wo in einem HTML -Dokument stehen darf, ist in der DTD festgelegt. Eine solche DTD kann entweder, vor allem bei kleinen DTDs, direkt in das

XML-Dokument eingebettet werden oder wie am Beispiel von HTML über den Ausdruck *<!doctype html public "-//W3C//DTD HTML 4.0 //EN">* bekannt gemacht werden.

Eine DTD könnte folgendes Aussehen für das XML-Beispiel auf Seite 17 haben:

```
1|  <!Element Dokument (Titel, Eintrag +)>
2|  <!Element Titel (#PCDATA)>
3|  <!Element Eintrag (Name, Vorname, Anschrift, E-Mail)>
4|  <!Element Name (#PCDATA)>
5|  <!Element Vorname (#PCDATA)>
6|  <!Element Anschrift (#EMPTY)>
7|  <!Element EMail (#PCDATA)>
8|  <!ATTLIST Anschrift
9|              Strasse CDATA #REQUIRED
10|           Plz CDATA #REQUIRED
11|           Ort CDATA #REQUIRED>
12|<!ATTLIST Titel Besitzer CDATA #REQUIRED >
```

Abbildung 6: DTD-Beispiel

In Abbildung 6:DTD-Beispiel sind die Elemente aus dem XML-Beispiel (Abbildung 4: XML-Beispiel) wieder zu finden, es kann die Struktur erkannt werden. In Zeile 1 findet sich das Root-Element „Dokument“, es beinhaltet einen „Titel“ und mehrere „Eintrag“-Elemente. Das Element „Titel“ kann aus beliebigen Zeichen bestehen, was in Zeile 2 durch den Ausdruck #PCDATA festgelegt ist. In Zeile 13 ist das zum „Titel“ gehörige Attribut „Besitzer“ definiert. Das Attribut „Besitzer“ ist vom Typ CDATA, kann also aus beliebigen Zeichenfolgen (beliebige Zeichen in beliebiger Reihenfolge) bestehen und muss angegeben werden, was durch #REQUIRED dargestellt ist. Die anderen Elemente folgen diesem Schema.

Durch Verwendung der DTD kann das XML-Dokument auf Validität überprüft werden, d.h. ob die Regeln der DTD bei der Erstellung des XML-Dokuments eingehalten wurden.

3.1.4 XML-Schema

Nachdem eine Möglichkeit ist, mittels DTD XML-Elemente, Attribute und deren Inhalte (#PCDATA, CDATA) durch eine nicht XML-Syntax zu beschreiben, kommt jetzt eine alternative Definitions-Möglichkeit - das XML-Schema. Beim XML-Schema wird zum Definieren der XML-Elemente, Attribute und Inhalte ebenfalls eine XML-Syntax verwendet. Des weiteren gibt uns XML-Schema die Möglichkeit Inhalte genauer zu definieren und das Vorkommen von Elementen genauer zu be-

stimmen. Im Vergleich zur DTD ist es mittels XML-Schema zusätzlich möglich die Anzahl der möglichen Elemente, Datentypen von Elementen und Attributen sowie eigene Datentypen zu definieren. Eine XML-Schema-Datei (*.xsd) könnte folgendes Aussehen für das XML-Beispiel auf Seite 17 haben:

```
1|  <?xml version="1.0" encoding="ISO-8859-1" ?>
2|  <xs:schema xmlns:xs="http://www.w3.org/2001/XMLSchema"
3|          targetNamespace="http://www.mydiploma.com"
4|          xmlns="http://www.mydiploma.com" elementFormDefault="qualified">
5|          <xs:element name="Dokument">
6|                  <xs:complexType>
7|                          <xs:sequence>
8|                                  <xs:element name="Titel" type="Titel"
9|                                                  minOccurs="1"  maxOccurs="1" />
10|                                 <xs:element name="Eintrag" type="Eintrag"
11|                                         minOccurs="0" maxOccurs="unbounded" />
12|                         </xs:sequence>
13|                 </xs:complexType>
14|         </xs:element>
15|         <xs:complexType name="Titel">
16|                 <xs:sequence>
17|                         <xs:element name="Titel" type="xs:string" />
18|                 </xs:sequence>
19|                 <xs:attribute name="Besitzer" type="xs:string" use="required" />
20|         </xs:complexType>
21|         <xs:complexType name="Eintrag">
22|                 <xs:sequence>
23|                         <xs:element name="Name" type="xs:string" />
24|                         <xs:element name="Vorname" type="xs:string" />
25|                         <xs:element name="Anschrift" type="AnschriftTyp" />
26|                         <xs:element name="EMail" type="xs:string" />
27|                 </xs:sequence>
28|         </xs:complexType>
29|         <xs:complexType name="AnschriftTyp">
30|                 <xs:attribute name="Strasse" type="xs:string" use="required" />
31|                 <xs:attribute name="Plz" type="xs:string" use="required" />
32|                 <xs:attribute name="Ort" type="xs:string" use="required" />
33|         </xs:complexType>
34| </xs:schema>
```

Abbildung 7: Beispiel XSD

In den Zeilen 2 bis 4 ist ein weiteres, bisher noch nicht erwähntes Element eines XML-Dokuments zu erkennen - ein so genannter Namensraum (name space). Über solche Namensräume (hier: www.mydiploma.com) wird es in XML möglich gleiche Elementnamen durch Verwendung des Namensraum Präfix zu unterscheiden. Dies ist vor allem dann von Interesse, wenn mehrere, voneinander unabhängige DTDs oder Schemata in einem XML-Dokument Verwendung finden. Es ist beispielsweise ein XML-Schema vorstellbar, in dem ein Adressbuch abgebildet wird. Hier würde

ein Element „Name“ verwendet um den Nachnamen von Personen zu definieren. Ein weiteres Schema bildet Fahrzeuge ab und hier würde ebenfalls ein Element „Name“ Verwendung finden, allerdings um die Typenbezeichnung zu definieren. Soll jetzt eine XML-Datei erstellt werden, in denen beide XML-Schemata Verwendung finden, so ist eine eindeutige Zuordnung nur noch durch das Präfix möglich.

Für XML-Schema gibt es eine Beschreibung, in der beispielsweise die Datentypen definiert werden, daher sind zahlreiche Datentypen (String, int, date, double, boolean, ...) bereits bekannt und können zum Aufbau von komplexen Datentypen (complexTypes Zeile 6, 14, ...) verwendet werden. Komplexe Datentypen können, wie im Beispiel zu erkennen ist, nicht nur aus einfachen Datentypen sondern ebenfalls aus komplexen Datentypen bestehen.

In Zeile 9 ist ein Beispiel für das Vorkommen von Elementen zu sehen, hier können direkt Zahlen angegeben werden, was es im Gegensatz zur DTD ermöglicht die Häufigkeit der Verwendung eines Elements exakt vorzugeben.

Nicht zu sehen ist die Möglichkeit Datentypen mit Einschränkungen (restrictions) zu versehen. Diese Einschränkungen können unter anderem in Form von regulären Ausdrücken angegeben werden, womit ein sehr mächtiges Werkzeug zur Verfügung steht, da diese Einschränkungen direkt beim Validieren überprüft werden können.

3.1.5 Formatierungssprachen

Nachdem nun die Basis von XML in den Grundzügen bekannt ist, stellt sich die Frage nach der Verwendung. Welchen Nutzen hat der ganze Aufwand der strukturierten Erfassung von Daten und wie kann eine Maschine diese Informationen verarbeiten?

Durch die Verwendung von Formatierungssprachen wird es möglich die Informationen für unterschiedliche Ausgabekanäle zu Formatieren. Ein Beispiel wäre die Ausgabe des XML-Dokuments in einer bestimmten Formatierung. Es kann im einfachsten Fall festgelegt werden, dass bestimmte Elemente beispielsweise in Fettschrift dargestellt werden sollen oder eine Tabellen-Struktur zur Darstellung genutzt wird. Es sind grob zwei Gruppen von Formatierungssprachen zu unterscheiden. Zum einen gibt es CSS (Cascading Style Sheets), die zur Laufzeit, also zum Zeitpunkt der Anzeige, eingesetzt werden. Das Originaldokument wird anhand der im CSS hin-

terlegten Anweisungen zur Anzeige gebracht. CSS sind vor allem durch den Einsatz in HTML weit verbreitet. Die zweite Gruppe mit XSLT (XSL-Transformation) und XPath (XML Path Language) dient im Gegensatz zu CSS nicht zum Formatieren während der Laufzeit, sondern zum Transformieren des XML-Dokuments in ein anderes Format. So kann aus dem XML-Dokument anhand einer XSLT-Formatierungsbeschreibung durch einen XSL-Prozessor beispielsweise ein PDF-Dokument erstellt werden. Durch die Verwendung von XSLT oder XPATH in Verbindung mit einem XSL-Prozessor können so aus einer XML-Datenquelle unterschiedliche Ausgabedokumente erstellt werden.

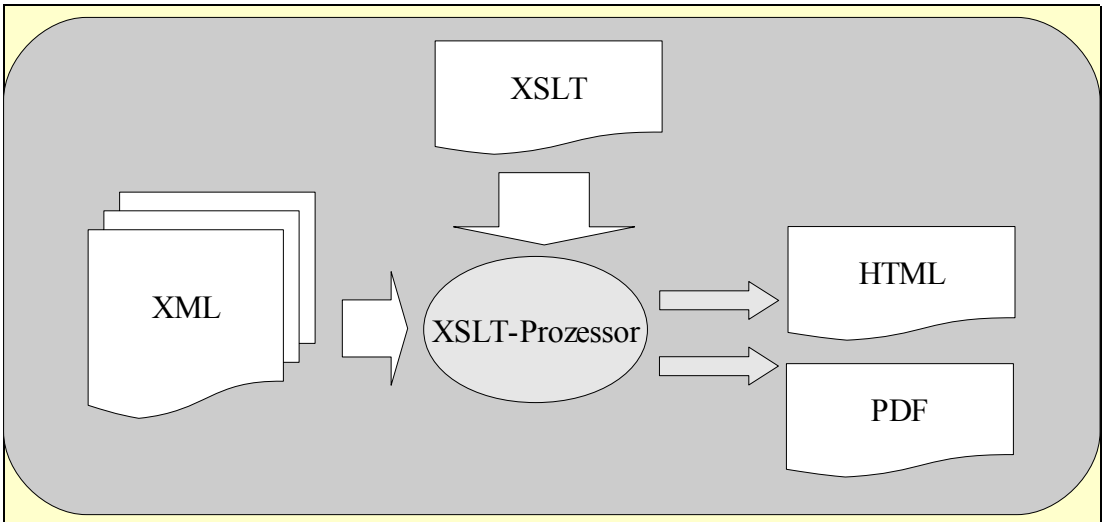


Abbildung 8: XSLT-Transformation

3.2 XForms

Bei XForms handelt es sich um eine Spezifikation des W3C-Konsortiums. XForms entstand aus einer Untergruppe der HTML-Arbeitsgruppe des W3C, die als eigene Aktivität ausgelagert wurde. Die XForms Spezifikation dient zur Beschreibung von Web-Formularen mit einer breiten Unterstützung unterschiedlicher Plattformen, vom Papier über Handhelds bis hin zu Desktop-Computern und Servern.

Im Gegensatz zu den in HTML verwendeten Formularen ist mittels XForms eine Trennung von Daten und Darstellung möglich. Für jede Plattform können zur Präsentation Adapter erstellt werden, die auf die jeweiligen Systemeigenheiten eingehen können.

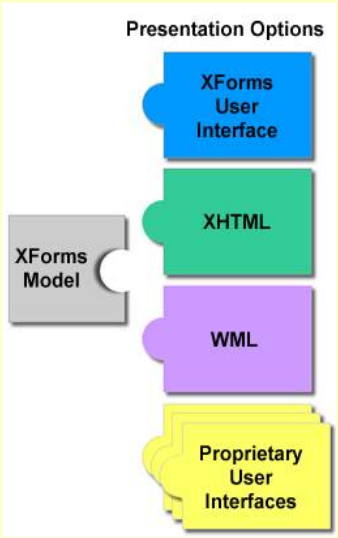


Abbildung 10: Darstellungsoptionen [W3C]

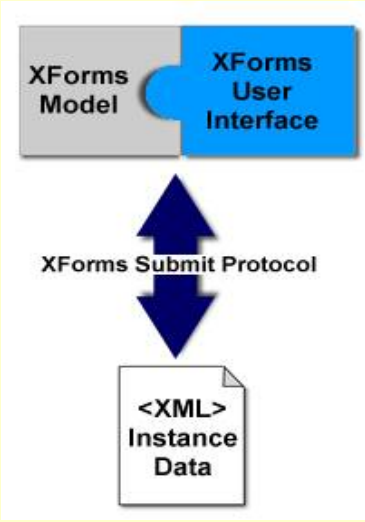


Abbildung 9: XForms Datenübermittlung [W3C]

Hieraus ergibt sich das in Abbildung 10: Darstellungsoptionen [W3C] dargestellte Baukastenprinzip, welches eine flexible und für das Web wichtige Systemunabhängigkeit bereitstellt. Mittels einer geräteunabhängigen XML-Definition, dem so genannten XForms Modell, können viele Standard- oder auch angepasste Benutzerschnittstellen angesprochen werden.

Die XForms-Benutzer-Schnittstelle stellt Standard-Bedienelemente bereit um beispielsweise die heute in XHTML (einer überarbeiteten Version von HTML) verwendeten visuellen Steuerelemente zu ersetzen. Es wird an unterschiedlichsten Benutzerschnittstellen gearbeitet. So gibt es Entwicklergruppen, die sich nicht mit der optischen Darstellung der Formulare, sondern mit der Sprachausgabe dieser beschäftigen. Im XForms Modell wird die Struktur der Daten festgelegt, wodurch eine maschinelle Verarbeitung aufgrund des XML-Formats möglich wird. Diese maschinelle Verarbeitung ist vor allem zum Automatisieren von Arbeitsabläufen sehr interessant, da es möglich ist Formulare automatisch (teil-) auszufüllen.

Im XForms-Übertragungsprotokoll (XForms Submit Protocol) wird schließlich festgelegt, über welche Übertragungskanäle die Formulardaten übermittelt werden können und welche Möglichkeiten bezüglich Pausieren und Wiederaufnehmen beim Ausfüllen des Formulars möglich sind. In Abbildung 10: Darstellungsoptionen

[W3C] sind die wichtigsten Aspekte von XForms noch einmal zusammengefasst dargestellt.

Die Schlüssel-Ziele von XForms sind:

- Unterstützung von Handhelds, TV- und PC-Browsern ebenso wie Drucker und Scanner
- Benutzerschnittstelle, die die Bedürfnisse von Firmen, Endverbrauchern und Gerätesteuerungs-Anwendungen gleichermaßen abdeckt
- Entkoppeln von Daten, Logik und Präsentation
- verbesserte Internationalisierung
- Unterstützung für strukturierte Formular-Daten
- mehrere Formulare pro Seite und mehrere Seiten pro Formular
- Unterstützung für Pausieren und Wiederaufnehmen
- nahtlose Integration durch andere XML-Elemente

[W3C]

3.3 XUL

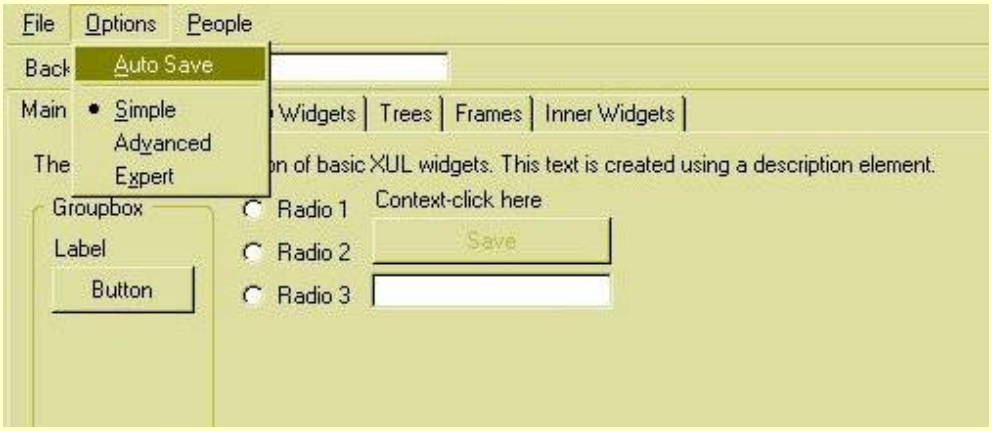


Abbildung 11: XUL-Beispiel [XULALLEM]

Mit dem XML-Dialekt XUL (XML User Interface Language) des Mozilla-Projekts [Moz] ist die Beschreibung von Oberflächen möglich. XUL entstand aus dem Mozilla-Projekt, um mit der hierfür entwickelten Gecko-Render-Einheit auch komplexe GUI[9]'s mittels XML gestalten und darstellen zu können. Hierdurch wurde ein systemunabhängiges Definitionsformat geschaffen, das sowohl die verwendeten grafischen Elemente, als auch deren Anordnung festlegt, aber die Freiheit lässt die Dar-

9 GUI - engl.Graphical User Interface- Grafische Benutzer -Schnittstelle/-Oberfläche

stellung der Elemente über Style Sheets zu ändern, um beispielsweise die Systemfarben für Fenster in einer Anwendung zu übernehmen. Die wohl bekannteste XUL-basierte Anwendung dürfte das Mozilla-Produkt Firefox sein. XUL-basierte Programme bestehen in ihrem Kern aus einer Wiedergabe-(Render-)Einheit (Gecko Rendering Engine), die anhand der in einem XUL-Dokument beschriebenen Elemente eine Oberfläche zur Anzeige bringt und die Verarbeitung von CSS (Cascading Style Sheets) mit übernimmt. Im Browser Firefox dient die Gecko Engine ebenfalls zur Darstellung von HTML-Seiten. XUL GUI-Elemente können mittels Javascript mit Logik und Funktionen versehen werden.

3.4 Web Services

Ein Web Service stellt Funktionen zur Kommunikation zwischen Server und Client bereit. Diese Art der Kommunikation erscheint bereits bekannt, da das Internet mit seinen unzähligen Webseiten ebenfalls nach diesem Prinzip funktioniert. Web Services stellen jedoch mehr eine Integrationsarchitektur zur Zusammenarbeit von Anwendungen dar, bei welcher Anwendungen Daten ohne menschliche Eingriffe miteinander austauschen können.

Web Services können in einem UDDI (Universal Definition, Discovery and Integration) -Verzeichnis registriert werden um von potentiellen Nutzern (besser) gefunden zu werden. Der Zusammenhang zwischen dem UDDI-Verzeichnis, dem Web Service Anbieter und dem Web Service Nutzer stellt Abbildung 12 dar.

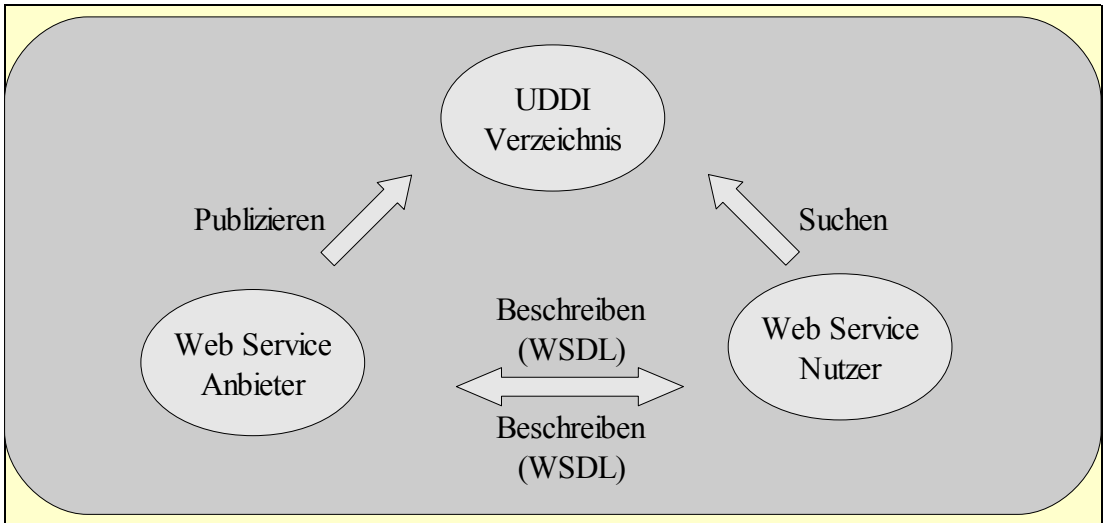


Abbildung 12: Übersicht UDDI [WSKOMP]

Ein Web Service wird durch eine WSDL (siehe 3.4.1 WSDL) beschrieben und tauscht Nachrichten beispielsweise über SOAP aus. Für die Kommunikation stehen unterschiedliche Wege zur Verfügung. So können Nachrichten mittels HTTP ebenso übertragen werden wie per E-Mail. Der große Vorteil gegenüber anderen Technologien verteilter Systeme liegt in dieser Datenübertragungsmethode. Bei Verwendung von Corba, DCOM, .NET-Remoting oder Java RMI als Basis der Kommunikation über Netzwerke hinaus stellt sich das Problem der Sicherheit. Meist sind (oder sollten) Firmennetzwerke durch Firewalls gesichert (sein). Diese Firewalls können Web Services aufgrund der Nutzung des HTTP-Protokolls ohne Konfigurationsaufwand passieren, da die Kommunikation nur über einen Port (Port 80) läuft und dieser für den normalen Arbeitsalltag oft schon freigeschaltet ist.

Bei Web Services kommen offene (frei nutzbare) Internettechnologien zum Einsatz, für die es bereits gute (Tool-) Unterstützung gibt und die weit verbreitet sind. Es können hierdurch bereits bestehende Systeme für die Nutzung von Web Services verwendet werden und Strukturen wie beispielsweise für die Lastverteilung (load balancing) genutzt werden.

Ein weiteres Merkmal ist die systemübergreifende Nutzbarkeit von Web Services. Bei verteilten Anwendungen unter Verwendung von Java RMI oder .NET-Remoting

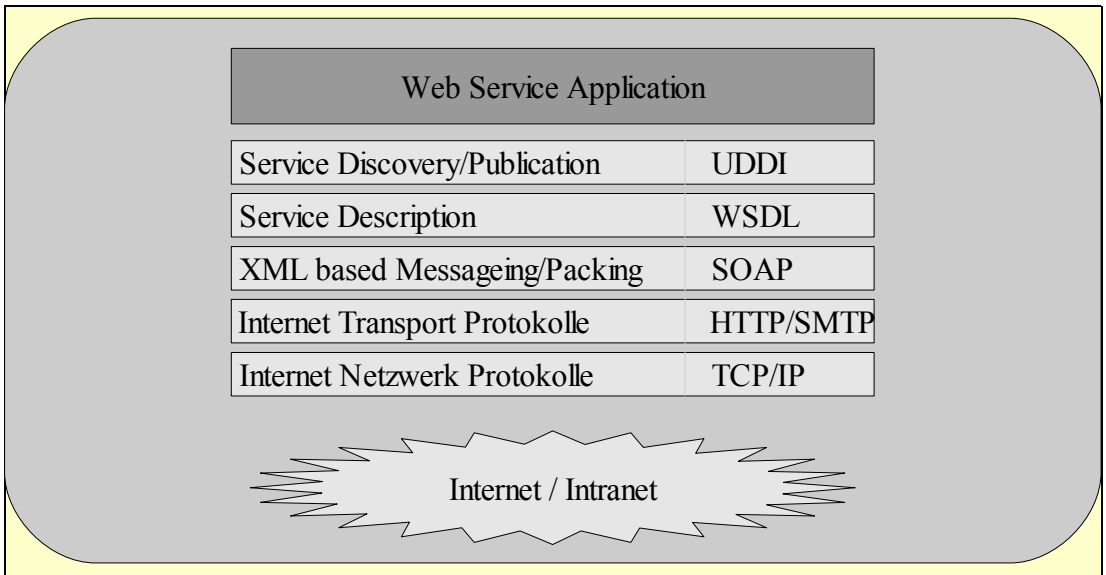


Abbildung 13: Web Service Protokollstapel [WSKOMP]

ist man auf bestimmte Programmiersprachen festgelegt, die auf Server- und Client-Seite identisch sein müssen. Eine Kommunikation von einem zum anderen System ist eher selten möglich. Bei Web Services besteht dieses Problem nicht (bzw. fast

nicht, es gibt trotz der bei Web Services verwendeten Standards unterschiedliche Interpretierungen durch die Hersteller, was zu Problemen führen kann), da allgemein anerkannte Standards verwendet werden und durch den Datenaustausch mittels SOAP, einem XML-Format (siehe 3.4.2 Soap), von Programmiersprachen unabhängige Kommunikation gewährleistet ist, können Verbindungen zwischen verschiedenen Systemen geschaffen werden.

3.4.1 WSDL

Die WSDL (Web Service Description Language) ist die Beschreibungsdatei eines Web Service. Sie verwendet ein XML-Format um einen Web Service mit den relevanten Daten zu beschreiben. Eine WSDL hat den Aufbau wie in Abbildung 14 dargestellt.

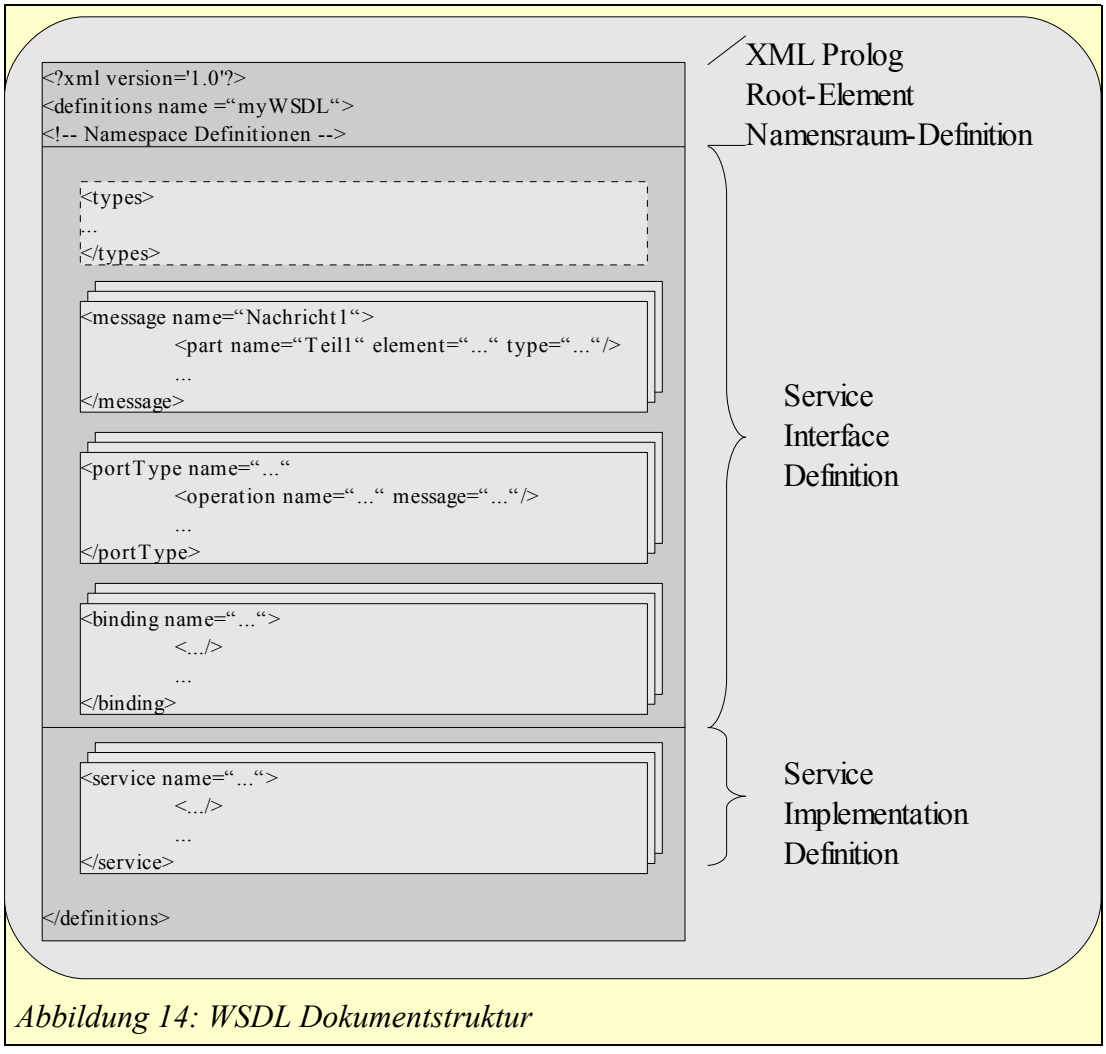


Abbildung 14: WSDL Dokumentstruktur

Der „definitions“ -Abschnitt stellt das root-Element dar. Er beinhaltet neben dem Namen des Web Service die Definition der verwendeten Namensräume sowie die übrigen Elemente.

In dem Abschnitt „types“ können neue Datentypen definiert werden, die für die Kommunikation genutzt werden sollen. Die WSDL unterstützt standardmäßig die Datentypen von XML Schema Definitionen (String, date, int,...), aus denen die neu definierten Datentypen zusammengesetzt werden. Datentypen aus XML Schema müssen nicht extra definiert werden.

In dem Abschnitt „messages“ werden die Nachrichten des Web Service beschrieben. Diese Nachrichten müssen entweder XML Schema Datentypen verwenden oder es muss eine Definition der verwendeten Datentypen in „types“ erfolgen.

Im Teil „portType“ werden die Ein- und Ausgabenachrichten, die während der Kommunikation ausgetauscht werden definiert. Nachrichten werden in diesem Element zu Operationen zusammengestellt. Hier wird definiert, welche Nachrichten (messages) für welche Operation zum Einsatz kommt. Die hier verwendeten Nachrichten müssen im Element „message“ definiert sein.

Im Teil „binding“ wird beschrieben, wie die Nachrichten über das Netzwerk übermittelt werden. Eine Anbindung an bestimmte Protokolle, wie SOAP erfolgt in diesem Element.

Der Teil „service“ fasst alle die Adressierung betreffenden Informationen zusammen. Es werden die Adressinformationen für die verwendeten Protokolle angegeben. Es sind IP-Adressen oder URLs angegeben, über die der Web Service erreicht werden kann.

Zwei weitere Elemente, „documentation“ und „import“ sind nicht in der Abbildung dargestellt, da sie nur optional sind. Diese Elemente werden nach dem Element „definitions“ in das Dokument eingefügt. Das „documentation“-Element dient zum Dokumentieren oder Kommentieren des Web Service in einer für Menschen lesbaren Form. Das „import“-Element erlaubt es externe Definitionen oder Nachrichten zu importieren. Hierdurch können auch Datentypen, die bereits definiert wurden, wieder verwendet werden und müssen nicht neu definiert werden.

3.4.2 Soap

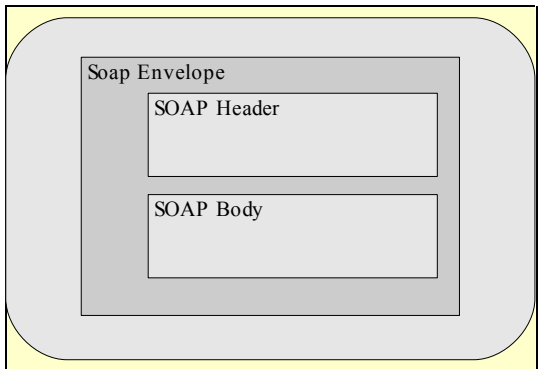


Abbildung 15: SOAP Nachricht

Das „Simple Object Access Protocol“ (SOAP) dient zum Dokumenttransport von einer Quelle zu einem Ziel. SOAP selbst ist ein auf XML basierendes Protokoll und es wird dazu genutzt sogenannte SOAP-Nachrichten zu erstellen.. Eine solche Nachricht besteht wie in Abbildung 15 dargestellt aus einem Umschlag, einem Kopf und einem Körper. Es gibt drei Arten von SOAP Nachrichten, den SOAP Request (Anfrage), die SOAP Response (Antwort) und den SOAP Fault (Fehler mit Beschreibung).

Der SOAP Umschlag (SOAP Envelope) stellt das XML root-Element dar. Im SOAP Envelope wird der Namensraum definiert. Für SOAP Version 1.0 lautet die Namensraum-URI „http://www.w3.org/2001/06/SOAP-envelope“.

Der SOAP Kopf (SOAP Header) ist, soweit verwendet, das erste Kind-Element des SOAP Envelope. Kind-Elemente des Header müssen ihre eigenen Namensräume definieren. Im Header können weitere Informationen über eine Nachricht untergebracht werden, müssen jedoch, wenn es nicht explizit angegeben ist vom Empfänger nicht verarbeitet werden können. Es ist beispielsweise möglich durch einen entsprechenden „Header“-Eintrag eine Authentifizierung notwendigerweise mit einzubinden. Sollte diese Authentifizierung vom Empfänger nicht verarbeitet werden können, wird eine Fehlermeldung zurückgegeben.

Der SOAP Körper (SOAP Body) beinhaltet die eigentlichen Nutzdaten. Auch hier muss, wie im Header, der Namensraum der Kind-Elemente festgelegt werden. Die Bezeichnung dieser Kind-Elemente entspricht dem Namen der aufzurufenden Methoden des Web Service. Die zum Aufruf notwendigen Attribute werden als Elemente (mit dem Attributnamen als Bezeichner) übermittelt. Handelt es sich um eine Antwort des Web Service, so wird das Ergebnis durch den Elementnamen „result“ gekennzeichnet. Im Falle eines Fehler wird ein Fault-Block übermittelt, der Informationen über den Fehler beinhaltet.

3.5 Mobilfunknetz

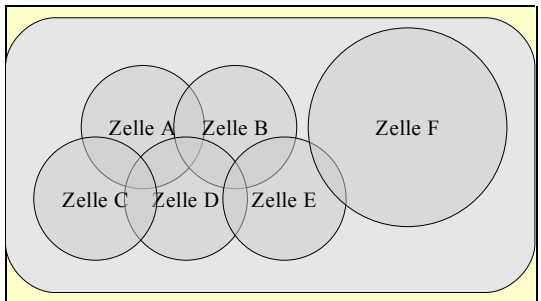


Abbildung 16: Mobilfunknetzzellen

GSM, GPRS und UMTS sind in Deutschland genutzte Funknetze. Diese Netze stellen die Grundlage für die Kommunikation mittels mobiler Endgeräte einem Großteil der Bevölkerung zur Verfügung. Die heute eingesetzten Netze sind in sich überlappende Zellen aufgeteilt und verwenden Protokolle um den reibungslosen Wechsel zwischen diesen Zellen zu gewährleisten. Durch diese Protokolle und die topologische Architektur des Netzes kann eine Zelle für das jeweilige Endgerät ermittelt werden, wodurch der Teilnehmer über seine festbleibende Nummer erreicht werden kann. Im Gegensatz hierzu musste bei den anfänglichen Mobilfunk-Netzen die Zelle des Teilnehmers bekannt sein, um ihn mittels einer Vorwahl erreichen zu können. Ein Wechsel von einer Zelle in eine andere Zelle hatte einen Verbindungsabbruch zur Folge.

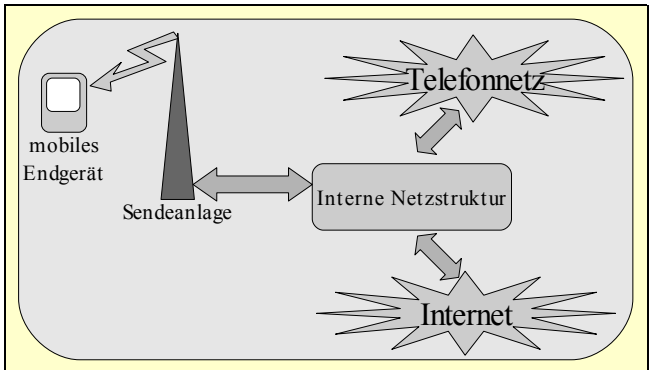


Abbildung 17: GSM Netzaufbau (vereinfacht)

In Abbildung 16 ist der Zellaufbau der Sendeanlagen symbolisiert dargestellt. Eine "GSM 900" (D-Netz) Zelle kann begründet durch den Frequenzbereich (Up-Link, vom mobilen Endgerät zur Sende-(Basis-)-Station, von 890 MHz bis 915 MHz sowie Down-Link, von der Basisstation zum mobilen Endgerät, von 935 MHz bis 960 MHz) in 127 Kanäle unterteilt werden. Durch die Anzahl der Kanäle wird die Anzahl der in einer Zelle angemeldeten Teilnehmer bestimmt. Die Sendeleistung nimmt mit der Größe der Zelle zu. Die Zellen A bis E stellen den Aufbau in dicht besiedelten Gebieten dar, es gibt viele kleine Zellen um einer große Anzahl von Endgeräten die Kommunikation zu ermöglichen. Die etwas abseits stehende große Zelle F symbolisiert weniger dicht besiedelte Gebiete. Zelle F kann einen größeren Bereich (bis zu

35 km bei GSM 900) abdecken. Da weniger Endgeräte gleichzeitig in diesem Bereich aktiv sein werden ist das dennoch ausreichend.

3.5.1 Java

Die Programmiersprache Java geht auf eine Entwicklung von Sun Microsystems im Jahr 1991 zurück. In Java wurden Konzepte und Elemente aus bestehenden Systemen übernommen und neu kombiniert. So wurde der Ansatz der virtuellen Maschine, die zwischen Programmcode und Hardware eine Zwischenschicht darstellt, von der Programmiersprache Smalltalk übernommen um unabhängig von der Hardware die geschriebenen Programme ohne erneutes Kompilieren verwenden zu können. Die Datentypen und Operationen stammen der Programmiersprache C++ ab.

Wichtige Eigenschaften von Java sind Einfachheit, Stabilität, Objektorientierung, Verteilbarkeit, Sicherheit und Portierbarkeit.

Eine höhere Einfachheit und Stabilität gegenüber C und C++ konnte durch das Weglassen von Konstrukten wie beispielsweise Zeigerarithmetik erreicht werden.

Die echte Objektorientierung, die in Java umgesetzt wurde lässt nur Methoden in Objekten zu, wodurch der objektorientierte Ansatz von den Programmierern umgesetzt werden muss.

Die Verteilbarkeit (der Einsatz in verteilten Systemen) wurde bereits zu Beginn der Entwicklung von Java durch eine umfangreiche Unterstützung von Netzwerkverbindungen und Zugriffsverfahren auf Verteilte Objekte mit bedacht. Durch das Konzept von Applets, also Programmcode, der über ein Netzwerk von einem Server zu einem Client übertragen und ausgeführt werden kann wurde ein sehr guter Verteilmechanismus für Software entworfen.

Die Sicherheit, die Java verwendet, schützt vor systemkritischen Zugriffen durch die Programme. Java bietet durch die Objektorientierung und den Wegfall von Zeigern, wie sie in C (++) verwendet werden, auch Schutz vor Speicherzugriffsverletzungen. Hierdurch können Programmteile und Variablen nicht versehentlich überschrieben und dadurch schadhafter Code eingeschleust werden.

Die Portierbarkeit galt bei der Entwicklung von Java als wichtiger Punkt, was durch den Ausdruck „write once, run anywhere“ deutlich geprägt wurde. Diese Portierbarkeit wurde durch den Einsatz der virtuellen Maschine erreicht. Hierbei handelt es

sich um einen Bytecode-Interpreter. Für jedes System muss eine solche virtuelle Maschine zunächst einmal vorliegen, dann können die Programme in Form von sogenanntem Bytecode in dieser virtuellen Maschine gestartet werden. Die Ein- und Ausgabe und andere Systemzugriffe übernimmt die virtuelle Maschine und das Programm greift auf die Ressourcen über die virtuelle Maschine, also nicht direkt zu. Der Bytecode stellt einen Zwischencode dar, der das Ergebnis des Kompilierens ist. (Andere Programmiersprachen erstellen durch das Kompilieren sogenannten Maschinencode, der jedoch prozessorspezifisch ist, dann allerdings direkt vom Prozessor verarbeitet werden kann.)

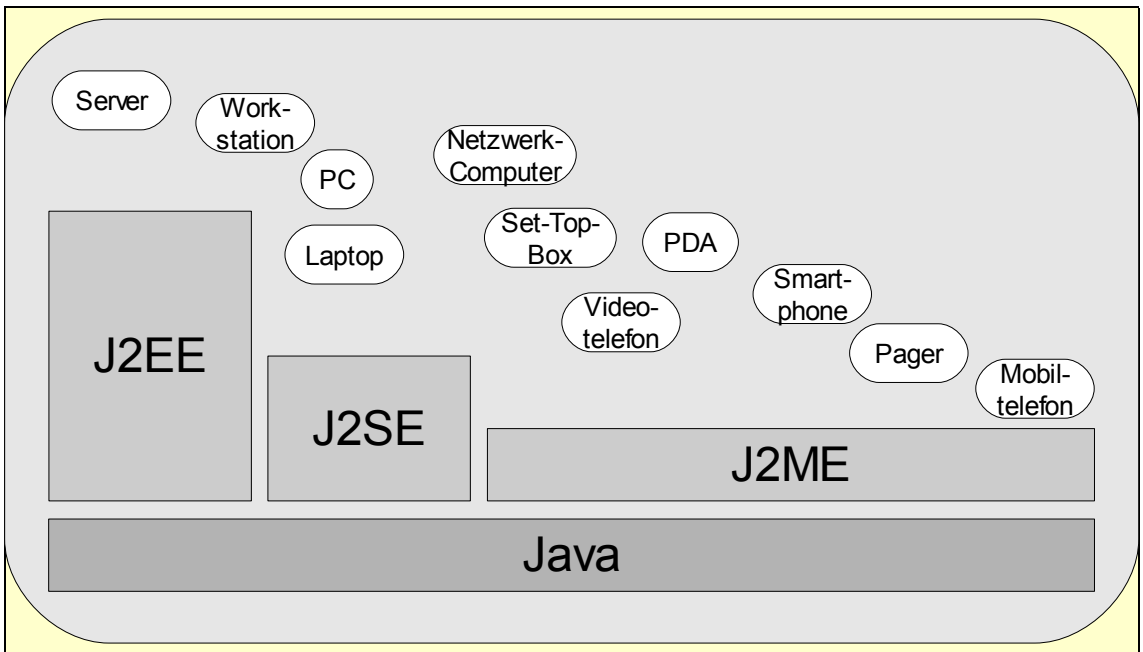


Abbildung 18: Java Anwendungsgebiete [J2MEKDS04]

Java ist in Pakete aufgeteilt, die als APIs (Application Programming Interface) verfügbar sind und die Klassenbibliothek darstellen. Eigene APIs können in Form von *.jar (Java Archive)-Dateien ebenfalls bereitgestellt und bei Bedarf importiert werden.

Aufgrund des weiten Anwendungsgebietes wurde Java aus Effizienzgründen für unterschiedliche Anwendungsbereiche aufgeteilt. Es entstanden die Teile Java EE, Java SE und Java ME. Java EE (Java Enterprise Edition) ist hierbei für den Einsatz im Serverumfeld ausgelegt und stellt das Größte der drei Pakete dar. Java SE (Java Standard Edition) wird im Bereich der Desktop-Computer, also Standard PCs eingesetzt. Java ME (Java Micro Edition) ist sehr klein gehalten was den Speicher betrifft und

wird im Bereich von Kleinstcomputern wie integrierten Systemen und Mobiltelefonen eingesetzt.

Die virtuelle Java Maschine (Laufzeitumgebung) sowie ausführliche Information sind unter www.java.sun.com verfügbar.

3.5.2 Java 2 Micro Edition (J2ME)

Die kleinste Java-Edition (J2ME) ist für Geräte mit geringer Leistungsfähigkeit ausgelegt. Die Geräte, die J2ME unterstützen haben meist folgende Eigenschaften:
- geringe Speicherkapazität bei persistenten(dauerhaften) und flüchtigen Speicher
- geringe Prozessor-(Rechen-)Leistung
- im mobilen Umfeld geringe elektrische Leistung durch Akku-/Batteriebetrieb
- einfache Bedienoberfläche
- keine permanente Netzwerkverbindung.

Obwohl J2ME von J2SE abstammt sind nicht alle Objekte und Methoden zu finden, da die Implementierung mit den zur Verfügung stehenden geringen Ressourcen der Zielgeräte nicht möglich ist. Um J2ME an die jeweilige Umgebung besser anpassen zu können und dabei auf die begrenzten Ressourcen Rücksicht zu nehmen wurde hier eine Architektur gewählt, die eine flexible, modular aufgebaute Laufzeitumgebung unterstützt.

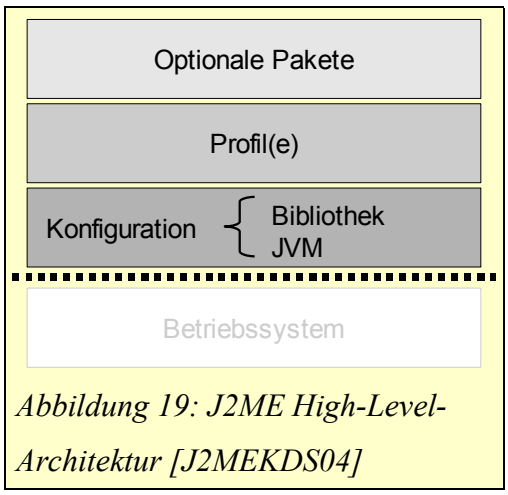


Abbildung 19: J2ME High-Level-Architektur [J2MEKDS04]

Zunächst wurden Konfigurationen geschaffen, die unterschiedliche Geräte in Klassen unterteilen. Diese Konfigurationen definieren den minimalen gemeinsamen Kern einer Klasse. Derzeit gibt es zwei solcher Konfigurationen. Die CLDC (Connected, Limited Device Configuration) besitzt hierbei wohl die größte Verbreitung, da sie im Bereich der Mobiltelefone eingesetzt wird. Die CDC (Connected Device Configuration) ist für den Einsatz in stationären Geräten, wie Set-Top-Boxen ausgelegt.

Den nächsten Baustein für die J2ME Umgebung stellen die Profile dar. Die Geräte-

klassen werden durch die Profile in Gerätetypen unterteilt, die auf spezielle Leistungsmerkmale er Endgeräte eingehen. Die Profile stellen eine gemeinsame Grundlage für die Gerätetypen bereit. Bei den mobilen Endgeräten der CLDC findet meist das Profil MIDP (Mobile Information Device Profile) Verwendung. Aktuell ist MIDP in Version 2 verfügbar und wird von aktuellen Geräten auch in Version 2 unterstützt.

Spezielle Erweiterungen, die sich von Gerät zu Gerät unterscheiden können, sind die optionalen Pakete. Meist bieten die Hardware-Hersteller eigene Erweiterungen an, wodurch die Kompatibilität einer Anwendung natürlich stark leidet.

Um eine gute Kompatibilität, unabhängig von Hersteller und Gerät, erreichen zu können werden Konfigurationen, Profile und optionale Pakete wie beispielsweise zur Bluetooth[10]-Unterstützung über so genannte JSRs (Java Specification Request) definiert. Diese JSRs werden von einer Hersteller übergreifenden Arbeitsgruppe (JCP- http://www.jcq.org) definiert und von den Herstellern bei Bedarf implementiert. Es ist bei der Entwicklung einer Anwendung hierdurch möglich optionale Pakete in Form von JSRs zu verwenden und die verwendeten JSRs in der Dokumentation der Anwendung zu notieren. Verwendet eine Anwendung JSRs die vom Gerät nicht unterstützt werden, ist diese Anwendung zu diesem Gerät nicht kompatibel, kann also dort nicht ausgeführt werden.

Anwendungen für das MIDP-Profil werden als MIDlets bezeichnet. MIDlets sind normale Java-Klassen, die von der abstrakten Klasse MIDLET der MIDP-Bibliothek abgeleitet sind und Methoden zum Lebenszyklus enthalten.

Die kleinste installierbare Einheit bilden MIDlet-Suites, die aus mindestens einem MIDlet bestehen. MIDlets einer Suite können Daten durch Variablen gemeinsam verwenden und besitzen gemeinsame Zugriffsrechte auf dauerhaft gespeicherte Daten.

Zum dauerhaften Speichern von Daten stehen in MIDlet-Suites so genannte „Record-Stores“ zur Verfügung, die durch das RMS (Record Management System) organisiert werden. Diese Record-Stores bieten die Möglichkeit Daten sicher vor Fremdzugriffen zu speichern, indem der Zugriff von anderen MIDlet-Suites beschränkt, bzw. ganz gesperrt werden kann.

10 Bluetooth ist eine Funkverbindung zum Datenaustausch mit einer Reichweite von ca. 10 bis zu 100 Metern.

Für die Oberflächengestaltung von Anwendungen stehen einige einfache grafische Elemente wie Dialogboxen oder Formulare zur Verfügung. Die Darstellung von grafischen Elementen kann sich abhängig vom Endgerät ändern. Grafische Elemente werden ebenso wie Steuerelemente an die jeweiligen Gerätes angepasst.

Zur Verarbeitung von XML stehen frei verfügbare Pakete zur Verfügung, die genutzt werden können. Eines dieser Pakete ist kXML. kXML beinhaltet sowohl einen DOM als auch einen SAX-Parser und ist an die J2ME-Umgebung bezüglich der knappen Ressourcen angepasst.

4 Realisierung der Aufgabenstellung

4.1 Konzeption und Planung

Zur Konzeption wurde zunächst die Struktur einer WSDL analysiert. Bedingt durch den Aufbau und das Funktionsprinzip eines Web Service wurde zur Darstellung auf dem mobilen Endgerät ein Formular gewählt.

Formulare werden von J2ME durch eine eigene Klasse repräsentiert und stellen einen guten Ansatz zur Eingabe von Daten dar. Zur Speicherung generierter Formulare wurden bereits auf dem Markt befindliche Standards zur Formular- und Darstellungsbeschreibung evaluiert.

Um Daten wie die Definition von Formularen und ausgefüllte Formulare persistent (dauerhaft) speichern zu können mussten hierfür Mechanismen in J2ME gefunden werden (mehr hierzu weiter unten).

Es ergab sich die in Abbildung 20 grob dargestellte Struktur für den Ablauf der

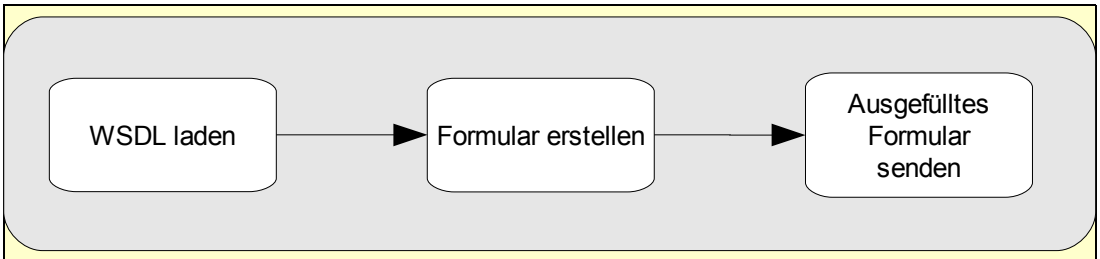


Abbildung 20: Grobe Darstellung des Programmablaufs

Anwendung.

Im Folgenden werden nach Vorstellung der Entwicklungsumgebung die einzelnen Bereiche der Anwendung anhand einer typischen Benutzung genauer vorgestellt.

4.2 Verwendete Tools und Programme

Für die Realisierung eines Prototypen ist die Verwendung einer entsprechenden Entwicklungsumgebung von großer Bedeutung. Fast alle Hersteller mobiler Endgeräte bieten für ihre Produkte Entwicklungswerkzeuge an. Diese sind jedoch oft nicht sonderlich gut. Es fehlen oft nützliche Unterstützungen, wie Autovervollständigen oder Fehlererkennungen, die man aus aktuellen Entwicklungsumgebungen für J2SE gewohnt ist. Hierdurch stellt das Arbeiten mit diesen Tools einen Rückschritt bezüg-

lich Komfort und Effektivität dar. Eine herstellerunabhängige Lösung bietet die Entwicklungsumgebung der Firma Sun. Sun bietet mit dem „Sun Java Wireless Toolkit“ (sjwtoolkit) eine Umgebung, die das Entwickeln und Testen von Anwendungen erlaubt und hierbei auf die MIDP Standardimplementierung zurückgreift. Durch das Einbinden von Gerätebeschreibungen und Emulatoren der Hersteller werden mit dem sjwtoolkit jedoch auch Entwicklungen für spezielle Geräte unterstützt. Das Arbeiten mit dem sjwtoolkit ist jedoch ebenfalls nicht sonderlich komfortabel.

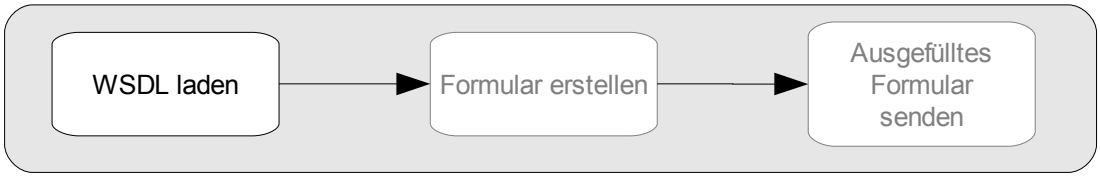
Eine gute Alternative zu den Tools der Hersteller stellt die Entwicklungsumgebung Eclipse[11] dar. Eclipse zeichnet sich vor allem durch seine flexible Erweiterbarkeit aus. Hierbei können Erweiterungen in Form von Plugins wie Bausteine hinzugefügt werden, die neue Funktionen bereitstellen. Es gibt eine Vielzahl solcher Erweiterungen, darunter ein Projekt mit dem Namen EclipseME[12]. Die Entwickler von EclipseME haben sich die Entwicklung einer J2ME-Entwicklungsumgebung zur Aufgabe gestellt und hierbei bereits sehr gute Erfolge erzielt. Bei EclipseME kommen, wie bei dem sjwtoolkit von Geräte-Herstellern bereitgestellte Beschreibungen und Emulatoren zum Einsatz.

Abbildung 21: Nokia 6230i

Für die Implementierung des Prototypen kam Eclipse in Version 3.2 mit dem EclipseME-Plugin Version 1.5 zum Einsatz. Zum Testen wurde auf den Nokia Emulator für das Nokia Mobiltelefon 6230i zurückgegriffen, da dieses auch als reales Gerät zur Verfügung stand und damit als reales Testsystem genutzt werden konnte. Um die Kompatibilität mit Geräten anderer Hersteller zu Testen kam das „mobility toolkit“ der Firma BenQ(/Siemens) zum Einsatz. Hiermit wurde ein Siemens S65 emuliert, um das Verhalten der Anwendung auf einem zusätzlichen System feststellen zu können.

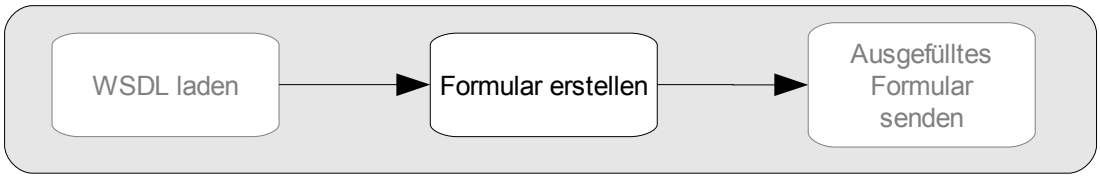
11 Eclipse entstand als Entwicklung durch die Firma IBM und wurde als open source zur Verfügung gestellt.[Eclipse]
12 EclipseME soll als Eclipse Plugin bei der Entwicklung von MIDlets helfen.[EclipME]

4.3 Details der einzelnen Pakete



Um einen Web Service nutzen zu können ist es zunächst notwendig die zu ihm gehörende WSDL zu laden. Der hierfür mögliche Weg über ein UDDI-Verzeichnis wird in dieser Arbeit nicht zum Einsatz kommen. Bei dem entwickelten Prototypen ist eine manuelle Eingabe der URL, die zur WSDL führt, vorgesehen. Zum Verwalten der bereits geladenen WSDLs kommt eine Art Historie zum Einsatz, die auf dem mobilen Endgerät gespeicherte WSDLs über ihre URL zur Auswahl bereitstellt.

Nachdem die Adresse (URL) einer WSDL auf dem mobilen Endgerät neu erfasst wurde, wird diese in der Historie hinterlegt und es erfolgt der Download dieser WSDL. Um erneutes Laden einer WSDL von der gleichen URL, und damit verbunden unnötige Datenübertragung (=Zeit und Kosten) zu vermeiden wird die WSDL im Speicher des Endgerätes persistent abgelegt. Hierfür wurde die Klasse „PersistentIN“ implementiert. Diese Klasse ist dafür zuständig angeforderte Daten zu liefern. Werden Daten, in diesem Fall eine WSDL von einer Quelle angefordert, sorgt die Klasse „PersistentIN“ für ein lokales Speichern, um bei einer erneuten Anforderung diese Daten schneller und kostengünstiger bereitstellen zu können.



Im nächsten Schritt wird das Formular für den Benutzer aufgebaut. In der Planung war zunächst vorgesehen die Beschreibung von Formularen in einem geeigneten Format persistent zu Speichern, um den Aufbau des Formulars für die Anzeige zu beschleunigen.

Nach Evaluierung der bereits verfügbaren Standards stellte sich XForms des W3C als geeignetes Format für die Speicherung des generierten Formulars heraus. Ebenfalls sehr interessant erschien XUL des Mozilla-Projektes. XUL wurde jedoch aufgrund seiner Mächtigkeit zur Darstellungsbeschreibung und der mangelnden Unterstützung für J2ME verworfen, könnte jedoch für Anwendungen im Windows-Mobile Bereich von Interesse sein.

Als die Ergebnisse der Implementierung getestet wurden, konnte die Performance des mobilen Endgerätes so positiv überraschen, dass der Ansatz eines speziellen Formats für die Speicherung der Formulardefinition verworfen wurde. Es erfolgte eine Implementierung, die jeweils zum Zeitpunkt der Darstellung das entsprechende Formular anhand der WSDL neu generiert. Ein nicht unerheblicher Nebeneffekt ist die Speicherersparnis, die aus dieser Vorgehensweise resultiert, da keine zusätzlichen Daten für die Definition von Formularen persistent im Gerät abgelegt werden müssen.

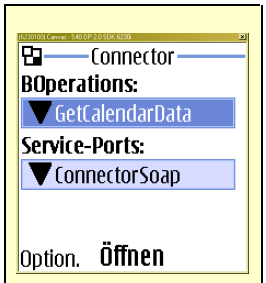
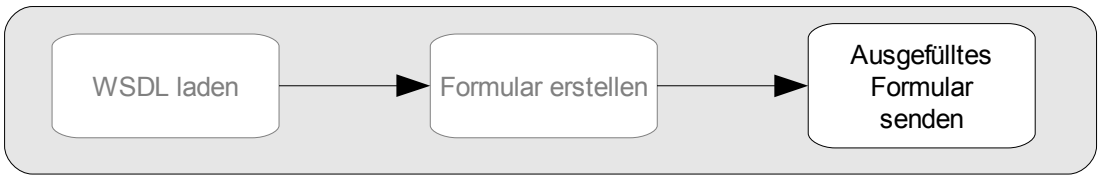


Abbildung 22: Auswahl über Auswahllisten

Aufgrund des Aufbaus eines Web Service startet das Formular mit der Auswahl des gewünschten Service. Die Struktur eines Web Service lässt sich sehr gut als Baumstruktur darstellen. Jedoch musste für das mobile Endgerät aufgrund der beschränkten Anzeigemöglichkeiten eine alternative Darstellungsform gefunden werden. Zum Einsatz kommen daher Auswahllisten („ChoiceGroups“), die den Anwender von der Wahl des Service über das Übertragungsprotokoll („port“) zur Auswahl der Operation führt. Nach Auswahl der Operation sind die notwendigen Eingabefelder in der WSDL eindeutig, da durch die Operation die Nachricht(en) („Message“) und damit die Eingabeparameter festgelegt ist (sind).

Im Folgenden, zur Anzeige gebrachten Formular werden die Eingabefelder dargestellt, die die eben ausgewählte Operation erfordert. Hierbei wird die Bezeichnung der Eingabefelder aus der WSDL-Types-Definition übernommen.

Der Benutzer hat nun die Möglichkeit das Formular auszufüllen und zu senden.



Während der Eingabe der Formulardaten kann es vorkommen, dass der Benutzer das Programm beenden will oder muss. Gründe hierfür können eingehende Telefonanrufe, zu tätigende Anrufe für Nachfragen oder auch andere sein. Um nach einem Neustart der Anwendung an diesem nur teilweise ausgefüllten Formular weiterarbeiten zu können ist es notwendig den Stand vor Beenden der Anwendung zu speichern.

Wird nach dem Ausfüllen des Formulars durch den Benutzer das Formular gesendet, wird zunächst versucht eine Verbindung zum Host des Web Service aufzubauen um die Formulardaten als SOAP-Nachricht zu übermitteln. Ist es nicht möglich eine Verbindung zu etablieren, werden die Formulardaten gespeichert um zu einem späteren Zeitpunkt einen erneuten Sendeversuch durchführen zu können.

Hierin besteht ein wesentlicher Vorteil des in dieser Arbeit verfolgten Ansatzes. Alternative Herangehensweisen an das Übermitteln von Daten, die auf einem mobilen Endgerät erfasst wurden, wären einfache HTML (oder WML)-Formulare. Hierbei müsste jedoch sowohl für das Aufrufen des Formulars als auch für das Übermitteln des ausgefüllten Formulars eine Internetverbindung bestehen. Steht beim Übermitteln des ausgefüllten Formulars die Internetverbindung (beispielsweise durch fehlende Funknetzabdeckung) nicht zur Verfügung werden die erfassten Daten mit hoher Wahrscheinlichkeit verloren gehen. Zumindest ist ein Weiterarbeiten ohne Datenverlust nicht möglich.

Bei erfolgreicher Übermittlung der Daten zum Web Service und Empfang des Ergebnisses durch das mobile Endgerät werden die empfangenen Daten auf dem Display des mobilen Endgerätes dargestellt.

4.3.1 Datenpersistenz

Bei Anwendungen, die auf erzeugte oder empfangene Daten auch nach einem Neustart der Anwendung zurückgreifen sollen oder müssen ist eine Möglichkeit diese Daten persistent zu speichern notwendig. Daten Persistent zu speichern bedeutet, dass diese dauerhaft auf einem geeigneten Speichermedium abgelegt werden. Im Umfeld der mobilen Endgeräte kommen hierbei aufgrund der geringen Größe meist Speicherkarten (SD-Karten, MMC-Karten) zum Einsatz. Desweiteren besitzen die Geräte einen nicht flüchtigen integrierten Speicher, der ebenfalls für Anwendungen zum Ablegen von Daten genutzt werden kann. Der interne Speicher ist bei der Großzahl der derzeit verfügbaren mobilen Endgeräte jedoch mit 1 MB bis 64 MB nicht sonderlich groß.

Ein MIDlet unterstützt zwei Arten Daten zu speichern.

Die erste Möglichkeit, die von allen MIDP-Umgebungen unterstützt wird, ist die Verwendung des RMS (Record Management System). Das RMS bietet die Möglich-

keit Daten in Form von byte-Arrays, einer primitiven Datenstruktur, in einem „RecordStore“ abzulegen. Hierbei besteht ein Problem darin, dass J2ME keine Unterstützung zum Serialisieren von Objekten besitzt und somit das Umwandeln der Objekte in byte-Arrays vom Programmierer übernommen werden muss. Die Daten liegen bei Verwendung von RecordStores im internen Speicher des Gerätes, wodurch sie vor fremden Zugriffen sicher sind.

Als zweite Möglichkeit Daten zu speichern steht die Unterstützung des JSR 75 zur Verfügung. Das JSR 75 (PDA Optional Packages for the J2ME Platform) ist in zwei optionale Pakete für mobile Endgeräte (nicht nur PDA) aufgeteilt, von denen eines den Zugriff auf Dateisysteme regelt. Durch die Unterstützung von Dateisystemen können Dateien erstellt und im Dateisystem des mobilen Endgerätes persistent gespeichert werden. Diese Dateien können dann auch auf einem Wechseldatenträger, der in das Gerät eingesetzt wurde, abgelegt werden. Bei manchen Geräten ist es sogar nur möglich Daten auf das eingesetzte Wechselmedium zu speichern. Daten die sich auf Wechseldatenträgern befinden sind jedoch vor Fremdzugriffen nicht mehr sicher geschützt, da der Datenträger entnommen und in einem anderen Lesegerät (beispielsweise an einem PC) ausgelesen werden kann.

Für die hier implementierte Anwendung kam die erste Variante also die Verwendung des RecordStores zum Einsatz. Für diese Entscheidung war neben der Datensicherheit die Kompatibilität ein wichtiges Kriterium. Es können um so mehr Endgeräte erreicht werden, je weniger optionale Pakete Verwendung finden. Das optionale Paket zum Zugriff auf Dateisysteme hat zum heutigen Zeitpunkt leider noch nicht die große Verbreitung unter den genutzten mobilen Endgeräten. In den kommenden ein bis zwei Jahren wird dies jedoch der Fall sein, da immer mehr aktuell erhältliche Geräte diese Pakete unterstützen.

Für die Realisierung der Datenspeicherung wurde in der Anwendung eine Klasse „Serializer“ implementiert. Diese dient zum Umwandeln aller Objekte und primitiven Datentypen[13] in byte-Arrays und ebenso in umgekehrter Weise aus den byte-Arrays wieder zurück in Objekte. Zum Serialisieren von Objekten werden diese in ihre Bestandteile (einfacheren Objekte) aufgeteilt, mit einer eindeutigen Kennzeichnung versehen und in ein byte-Array umgewandelt. Durch den eindeutigen Kenn-

13 primitive Datentypen bei Java ME sind :boolean, char, byte, short, int, long, float,double;

zeichner ist es möglich aus den byte-Arrays wieder die entsprechenden Objekte zu generieren.

Ein kurzes Beispiel anhand des Objektes String:

[Kennung] [Gesamtlänge des Arrays] [Stringdaten als byte-Array]

Die Kennung dient der eindeutigen Identifizierung und die Angabe der Länge ist notwendig um das Ende des Arrays bestimmen zu können. Nach diesem Prinzip, mit Angabe von Typ, Länge und Daten wurde für alle verwendeten Objekte die Serialisierung umgesetzt. Hierdurch wurde es möglich ein gesamtes Formular mit all seinen Felder zu serialisieren um es in einem RecordStore als byte-Array ablegen zu können.

Für den Zugriff auf die Daten des RecordStores bietet dieser selbst nur wenige Möglichkeiten, die ein einfaches Verwalten der Daten nicht gestatten. Aus diesem Grund wurde zum Verwalten der abgespeicherten Daten eine Verwaltungsstruktur implementiert, die es erlaubt ähnlich wie in einem Verzeichnis Daten anhand eines Namens innerhalb eines RecordStore schnell aufzufinden, auszulesen oder auch zu löschen. Hierfür wurde ebenfalls ein RecordStore verwendet, der mit den notwendigen Informationen zum Auffinden der Daten versehen wird.

4.3.2 Eingesetzte Fremdpakete

Für die entwickelte Anwendung ist es durch die Verwendung von Web Services notwendig XML zu verarbeiten. In J2ME MIDP2 ist jedoch keine Unterstützung für XML vorhanden. Es existiert zwar mit dem JSR172 (J2ME Web Services Specification) ein optionales Paket zur Unterstützung von Web Services auf J2ME-Geräten, jedoch ist dieses optionale Paket bisher erst bei neuen Modellen von mobilen Endgeräten zu finden. Es besteht demnach noch keine große Verbreitung unter den derzeit genutzten Endgeräten, wodurch dieses optionale Paket für diese Anwendung, die zu den aktuellen Geräten bestmögliche Kompatibilität gewährleisten soll, nicht zum Einsatz kommt.

Es fanden sich jedoch Alternativen in Form von kXML und kSOAP. Dieses sind zwei frei verfügbare Pakete, die Unterstützung zum Arbeiten mit Web Services liefern. Beide dieser Pakete sind für die Verwendung in J2ME und damit für mobile Endgeräte optimiert. kXML ist ein XML-Parser, mit dessen Hilfe ein XML-Dokument in die einzelnen Elemente und Attribute „zerlegt“ werden kann um darauf zu-

zugreifen. kXML bietet sowohl einen SAX- als auch einen DOM-Parser, von denen der DOM-Parser in dieser Arbeit zum Parsen der WSDL zum Einsatz kam. kSOAP stellt Funktionen zum Senden und Empfangen von SOAP-Nachrichten zur Verfügung.

Die beiden Pakete wurden in diese Arbeit integriert und werden als Teil der Anwendung auf die mobilen Endgeräte verteilt.

Auf weitere Fremdpakete, sowie auf optionale Pakete konnte bei der Implementierung der Anwendung verzichtet werden. Hierdurch wird eine sehr gute Kompatibilität zu den derzeit eingesetzten mobilen Endgeräten erreicht.

5 Fazit

Das Ziel der Arbeit, beliebige Web Services mit einer Anwendung nutzen zu können wurde erreicht. Es ist mit dieser Anwendung möglich Web Services, die zum Zeitpunkt der Implementierung dieser Anwendung noch nicht bekannt waren, zu nutzen.

Einschränkungen sind jedoch bei der Komplexität der verwendeten Web Services zu erwähnen, da sich zu komplexe Web Services an dem meist recht kleinen Display der Endgeräte nicht übersichtlich genug darstellen lassen. Des weiteren ist die Möglichkeit der Eingabe über das Zahlenfeld eines Mobiltelefons nicht für die Eingabe größerer Datenmengen geeignet wenn keine externe Tastatur angeschlossen werden kann.

Ebenfalls stellt die Verwendung der Web Services zum Teil ein Problem dar. Es ist anhand der Bezeichnungen der Felder nicht immer deren klare Verwendung und Bedeutung eindeutig für den Endbenutzer erkennbar. Hierfür bietet die WSDL jedoch Möglichkeiten der Beschreibung, die allerdings nur selten genutzt sind.

Es kann insgesamt festgestellt werden, dass vor allem kleine Web Services sehr gut mit dieser Anwendung genutzt werden können. Für komplexere Web Services werden speziell auf den Service angepasste Anwendungen sicherlich die bessere Wahl darstellen.

6 Ausblick

Für die Zukunft kann sicherlich das Potential der Anwendung noch gesteigert werden.

Es wäre eine Erweiterung bezüglich der Einbindung von Verzeichnisdiensten für die Zukunft erstrebenswert. Es müsste jedoch für die angebotenen Web Services eine für Menschen (besonders für Endbenutzer) gut verständliche Beschreibung der Einzelnen Services vorliegen um ein komfortables Arbeiten ermöglichen zu können.

Durch neuere Generationen mobiler Endgeräte werden sich neue Möglichkeiten ergeben. Die Darstellungsmöglichkeiten auf den Endgeräten werden sich durch größere Displays verbessern und komplexere Logiken zum Erzeugen der darzustellenden Formulare werden durch gesteigerte Verarbeitungsleistung (Rechenleistung) der eingesetzten Prozessoren neue Möglichkeiten eröffnen.

Es wäre denkbar zur Gestaltung einer Oberfläche auf eine XUL-Beschreibung zurückzugreifen, die im Web Service dann referenziert ist.

7 Anhang

7.1 Abkürzungsverzeichnis

API
: Application Programming Interface

CDC
: Connected Device Configuration

CLDC
: Connected, Limited Device Configuration

DOM
: Document Object Model

DTD
: Document Type Definition

GPRS
: General Packet Radio Service

GSM
: Global System for Mobile Communications

HTML
: Hypertext Markup Language

HTTP
: Hypertext Transfer Protocol

J2ME
: Java 2 Micro Edition

J2SE
: Java 2 Standard Edition

JSR
: Java Specification Request

kSOAP
: kilobyte SOAP

kXML
: kilobyte XML

MDA
: Mobile Digital Assistant

MIDlet
: Anwendung für J2ME MIDP-Profil

MIDP
: Mobile Information Device Profile

MMC-Karte
: Multi Media Speicherkarte

PDA
: Personal Digital Assistant

RMS
: Record Management System

SAX

SIMPLE API FOR XML

SD-KARTE
SECURE DISC SPEICHERKARTE

SOAP
SIMPLE OBJECT ACCESS PROTOCOL

URI
UNIFORM RESOURCE IDENTIFIER

URL
UNIFORM RESOURCE LOCATOR

WML
WIRELESS MARKUP LANGUAGE

WSDL
WEB SERVICE DESCRIPTION LANGUAGE

XML
EXTENSIBLE MARKUP LANGUAGE

XUL
XML USER INTERFACE LANGUAGE

7.2 Literaturverzeichnis

[EclipME] EclipseME Project home; http://eclipseme.org/

[Eclipse] Eclipse.org home; www.eclipse.org

[J2MEKDS04] Klaus-Dieter Schmatz: Java 2 Micro Edition. dpunkt.verlag, Heidelberg, 2004

[Moz] Home of the Mozilla Project; http://www.mozilla.org/

[RRZN01] RRZN/Universität Hannover: . Herdt-Verlag für Bildung, , 2004

[SYMOS] Symbian OS; www.symbian.com

[W3C] ; http://www.w3c.org

[WSKOMP] Michael Kuschke, Ludger Wölfel: Web Services kompakt. Spektrum, Akad. Verlag, Heidelberg, Berlin, 2002

[WSS04] Tobias Hauser, Ulrich M.Löwer: . Galileo Pres GmbH, Bonn, 2004

[XULALLEM] Übersicht aller XUL-Elemente; http://xulplanet.mozdev.org/tutorials/xultu/examples/allelem/allelem.xul