Hochschule für Angewandte Wissenschaften Hamburg
*Hamburg University of Applied Sciences*

***Diplomarbeit***

# Entwicklung CORBA-basierter Middleware für mobile Anwendungen

vorgelegt von
***Piotr Wendt***
am 11. Juni 2004

Studiengang Softwaretechnik
Betreuender Prüfer: Prof. Dr. Kai von Luck
Koreferent: Prof. Dr. Gunter Klemke

**Fachbereich Elektrotechnik und Informatik**
**Department of Electrical Engineering and Computer Science**

# Entwicklung CORBA-basierter Middleware für mobile Anwendungen

**Stichworte** Mobile Multimedia Middleware, Multimedia Dienste, Spontanes Vernetzen

## Zusammenfassung

Das Rechnen unterliegt einem Wandel. So werden in der näheren Zukunft immer mehr verschiedenartige mobile Geräte zum Einsatz kommen. Diese werden im mobilen Zustand auf die allgegenwärtigen Netzwerke zugreifen können, welche unterschiedliche Dienste nach außen bereitstellen. Hierdurch eröffnen sich neue Möglichkeiten für die Datenverarbeitung und die damit verbundenen Geschäftsprozesse. Gleichzeitig kommen aber neue Probleme zum Vorschein. Eines dieser Probleme ist die Mobilität selbst. So wird in dieser Arbeit dieses Problem anhand einiger Beispielszenarien näher analysiert. Dabei werden bestehende Wege und Möglichkeiten aufgezeigt, wie eine Middleware für eine dienstorientierte Architektur entwickelt werden kann. In diesem Zusammenhang werden verschiedene Technologien betrachtet. Hierbei gewonnene Erkenntnisse werden dann bei dem Entwurf einer möglichen Plattform für mobile Geräte verwendet. Die dabei prototypisch implementierte Middleware soll dann den mobilen Zugriff für mobile Geräte bereitstellen, wobei die eigentliche Kommunikation auf einer dienstorientierten Ebene durchgeführt werden soll.

# Development CORBA-based Middleware for mobile Applications

**Keywords** Mobile Multimedia Middleware, Multimedia Services, Spontaneous Networking

## Abstract

Due to the presence of mobile devices, computing paradigms are changing. On one hand, wireless network based services will offer new potentials of computing and business process handling. On the other hand, these services will also cause new problems. One of the core problems is that mobile devices are only part time connected. Based on sample scenarios, this thesis suggests extensions to CORBA based middleware which cover these problems. Furthermore, the benefits and limitations of the proposed solution will be discussed.

# Danksagung

An dieser Stelle möchte ich mich bei meinen betreuenden Professoren Herrn Prof. Dr. Kai von Luck und Herrn Prof. Dr. Gunter Klemke für ihren vielseitigen fachlichen Rat bedanken. Prof. Dr. von Luck möchte ich meinen besonderen Dank aussprechen der mich in jeder Phase meiner Arbeit sehr sachkundig, richtungsweisend begleitete und viel Geduld zeigte. Im Weiteren möchte ich mich bei denjenigen bedanken, die mich durch irgendeine Art unterstützt haben. Insbesondere danke ich André Lüpke, Michael Knop und Marco Bresch. Zum Schluss möchte ich mich bei meinen Eltern bedanken. Ohne sie wäre ich nie so weit gekommen.

# Inhaltsverzeichnis

# Abbildungsverzeichnis

# Tabellenverzeichnis

# Kapitel 1

# Einleitung – Welches Ziel wird verfolgt

„Wer glaubt, den Überblick über alles zu haben, leidet höchstwahrscheinlich an einem Mangel an Durchblick.“

Ernst Ferstl

## 1.1 Einleitung

Seit geraumer Zeit hört man in jedem Munde Stichworte wie „spontanes Vernetzen“, „mobiles Rechnen“, „Multimedia Dienste“, „mobiles Internet“. Diese und weitere Begriffe entstanden, als der Mensch angefangen hat, mehr unterwegs zu sein und dadurch stärker auf die Verfügbarkeit von Hilfsdiensten, während der Fortbewegung, angewiesen zu sein (Mobilität). Um auf diese zugreifen zu können, muss eine Netzwerkinfrastruktur gegeben sein, die diese Kommunikation ermöglicht. In unserem Informationszeitalter, sind die Computernetzwerke bereits allgegenwärtig. So umgeben uns die Funknetzwerke stattlicher, kommerzieller und privater Organisationen. Als bestes Beispiel kann hier das Internet erwähnt werden. Diese haben als Aufgabe, darin verfügbare Ressourcen nach außen allen, für die gemeinsame Nutzung, verfügbar zu machen. Dabei muss an dieser Stelle betont werden, dass Ressourcen als Synonym für alle möglichen Software- und Hardwarekomponenten stehen. Durch die heute anbietende Netzwerkinfrastruktur und durch den Bedarf nach Allgegenwärtigkeit verschiedener Dienste, entstand eine neue Sichtweise auf das Rechnen. Dieses hatte sich zu mobilen Dienstleitungen gewandelt, welche für den Benutzer global zur Verfügung stehen sollten. In diesem Zusammenhang entstand der Begriff der mobilen Multimedia Dienste.

In der Studie vom Institut für Zukunftsstudien und Technologien (IZT) (IZT u. a., 2001) wird weiter auf den Begriff der mobilen Multimedia Dienste aufgebaut. So werden die neuen Möglichkeiten der verteilten, mobilen Datenverarbeitung immer mehr Einfluss auf die Geschäftsprozesse der Unternehmen und Privatpersonen nehmen. Diese Änderung der Geschäftsprozesse, bzw. ihre Optimierung durch die neuen Möglichkeiten, äußert sich bereits heute. Beispielsweise gehen immer mehr Unternehmen dazu über, ihre Mitarbeiter mit mobilen Geräten auszustatten, um somit deren Verfügbarkeit und Mobilität zu steigern, wodurch

diese sich eine höhere Effizienz in Hinsicht der Zeit- und materieller Kosten erhoffen. Aus ähnlichen Gründen, also Zeit- und Geldersparnis, ist eine vergleichbare Entwicklung in dem privaten Sektor zu erwarten. Durch die neue Sichtweise werden außerdem bestimmte Geschäftsmodelle, wie Mikropayment erstmals realisierbar. Für deren Akzeptanz ist es aber letztendlich wichtig, ein vielfältiges Spektrum von angebotenen Diensten zur Verfügung zu stellen, wobei die Dienstleistungen stark die neuen Möglichkeiten der Mobilkommunikation, wie Orts- und Situationsabhängigkeit, nutzen sollten. Anwendungsfelder für mobile Multimedia Dienste sind zahlreich:

- Informationsdienstleistungen
- Unterhaltung
- Gesundheit- und Wellness
- Shopping
- Finanzdienstleistung
- Lernen
- Local Based Services (LBS)
- ...

Diese neue Sichtweise eröffnet neue Möglichkeiten für viele Bereiche unseres alltäglichen Lebens, bringt aber ebenfalls neue Probleme mit sich, die neue Überlegungen benötigen.

## 1.2 Aktueller Stand: Mobile Multimedia

Seit geraumer Zeit findet die Mobilfunktechnologie immer größere Akzeptanz. Diese hat sich inzwischen nicht nur in den kommerziellen Bereichen etabliert. Durch den immer attraktiver werdenden Preis ist diese Technologie inzwischen genauso weit im privaten Bereich verbreitet. Durch die große Verbreitung und immer größer werdende Bandbreite der Netztechnologien, eröffnen sich heute neue Möglichkeiten für die Bereitstellung neuer Dienste, welche auf diese Technologien aufbauen.

Neben der großen Akzeptanz, ist ebenfalls ein Wandel in dem Bereich der Kommunikationstechnologien zur verzeichnen. Immer mehr Endanwender besitzen heute neben dem Handy zusätzlich ein leistungsstärkeres Endgerät wie: PDA, Notebook und andere vergleichbare Geräte. Im diesem Zusammenhang ist ein Rückgang des Übertragungsvolumens der Sprache, auf der anderen Seite ein Zuwachs der zu übertragenden Daten zu verzeichnen. Diese Tatsache deutet darauf hin, dass eine Tendenz der Endanwender zu multimedialen

Diensten besteht. Diese Tendenz ist nicht nur wegen der höheren Bandbreite festzustellen, diese wird ebenfalls durch die neuen Endgeräte begünstigt. Die oben erwähnten Geräte zeichnen sich durch eine bessere Benutzerfreundlichkeit aus, die sich durch verbesserte Displays, Eingabemöglichkeiten, Performance und Leistungsumfang auszeichnen.

Schon heute sind einigen Multimedia Dienste in der Praxis vorzufinden. Diese unterstützen unseren beruflichen sowie privaten Alltag. So kann man bereits heute, bei einem entsprechenden Anbieter, ein Informationsdienst abonnieren. Dieser kann Informationen aus verschiedenen Bereichen liefern, die abhängig von dem angelegten Interessenprofil sind. Die Informationsversorgung kann asynchron per *Short Message Service* (SMS) oder On Demand per *Wireless Application Protocol* (WAP) durchgeführt werden. Im Finanzsektor haben sich die Multimedia Dienste heute am weitesten verbreitet. Im Mobile-Payment Bereich gibt es heute Dienste, mit denen man beispielsweise die Rechnung für eine Taxifahrt direkt mit dem mobilen Gerät begleichen kann. Es existieren darüber hinaus Dienste, die Bezahlung für ortsabhängige Dienstleistungen ermöglichen. Wertpapiertransaktionen, wie Kauf/Verkauf, bzw. Kursüberwachung haben sich im beruflichen Leben auch zunehmend etabliert.

Heute sind die ortsbasierten Dienste noch nicht so stark vertreten. Diese kommen vereinzelt in Bereichen vor, bei dem als Beispiel die Auffindung bestimmter ortsnaher Dienste benötigt wird. So kann hier das Auffinden des nächsten Zigarettenautomaten, oder das Abrufen der ortsbezogenen Verkehrsinformationen, wie Umleitungsvorschläge, genannt werden.

Viele Unternehmen vergrößern die Zahl der Außendienstmitarbeiter, um einen noch besseren Kunden-Vorort-Service als Zusatzleistung anbieten zu können. Hierbei entsteht ein Problem der Koordination diese Mitarbeiter. Dieses wird heute von einigen Unternehmen bereits erfolgreich mithilfe der mobilen Multimedia Diensten bewältigt. Ähnliche Beispiele kann man in den Speditionsfirmen vorfinden. Auch bei diesen wird für das Flottenmanagement die Funktechnologie erfolgreich genutzt.

In der Einleitung wurde der Begriff der *mobilen Multimedia* eingeführt, ohne diesen genau zu definieren. Die Abschnitte 1.2.1 und 1.2.2 sollen das Thema aufgreifen und näher betrachten. Dazu werden die nötigen Konzepte angesprochen und diskutiert. Am Ende sollte eine Definition für mobile Multimedia gefunden werden, die als Referenz für diese Arbeit dienen kann.

### 1.2.1 Ubiquitous computing

Der Begriff *Ubiquitous Computing*, zu Deutsch allgegenwärtiges Rechnen, der für eine neue Computer Sichtweise steht, wurde von Mark Weiser in (Weiser, 1991a) das erste Mal geprägt. Bei der weiteren Betrachtung von *Ubiquitous Computing* in Weiser (1993) wurde der Begriff noch stärker definiert. In diesem verstand Mike Weiser unter *Ubiquitous Computing*:

„*the method of enhancing computer use by making many computers available throughout*

*the physical environment, but making them effectively invisible to the user.*“

Mit anderen Worten, *Ubiquitous Computing* repräsentiert eine Umgebung, in der nicht direkt oder gar nicht erkennbare Rechner in allen erdenklichen Gegenständen zu finden sind. Deren Kompetenzen sind für einen speziellen Aufgabenbereich beschränkt und somit in ihrer Handhabung einfach und intuitiv. Daneben können diese in der unmittelbaren Umgebung befindlichen Geräte automatisch wahrnehmen, mit denen Informationen austauschen und diese entsprechend verarbeiten. D.h., jedes Gerät kann Wissen über seine Umgebung sammeln und somit diese wahrnehmen. Dementsprechend kann dieser auf diese Gegebenheiten reagieren, wodurch sein Verhalten ortsabhängig wird. Das Ganze geschieht ohne menschlichen Eingriff. Dieses macht das Verhalten der Geräte interaktiver und somit intelligent.

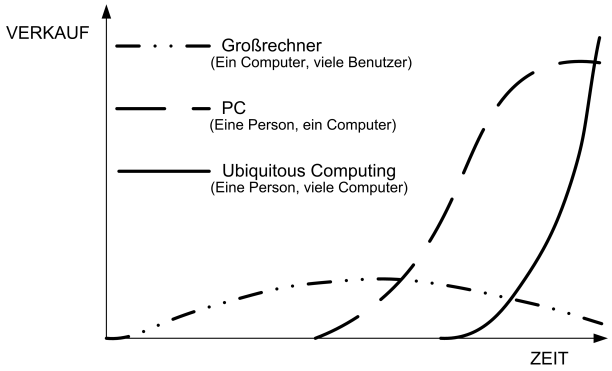


Abbildung 1.1: Der Trend bei der Datenverarbeitung (Mark Weiser, 2004).

Zurzeit der Begriffsbildung von *Ubiquitous Computing*, war dieses eher utopischer Natur, da die hierfür notwendigen Technologien nicht zur Verfügung standen. Seit dem hat sich eine Menge getan. Auf dem Markt erschienen neue Technologien, die bei der technischen Umsetzung, der von Mike Weiser aufgestellten Idee, hilfreich sein könnten. Dieser hatte bereits zu seiner Zeit die These über die Evolution der Rechner aufgestellt, welche besagt, dass man von den Großrechnern zu Ubiquitous Computing übergeht. Sie ist heute zum größten Teil Realität geworden. Die Tendenz, wie sie in der Abbildung 1.1 zu erkennen ist, geht in Richtung Ubiquitous Computing. So können Geräte mit den neuen Funktechnologien wie Bluetooth, WLAN und RFID auffindbar gemacht werden. Des Weiteren gibt es heute eine große Vielfalt von Rechnern, die klein genug sind und trotzdem ausreichend Rechenleistung für die anstehenden Aufgaben besitzen. Bei der Entwicklung der darauf laufenden *verteilten Systemen* hat sich ebenfalls viel getan. *J2EE*, *.NET*, *CORBA* sind Technologien, die verteilte Systeme unterstützen sollen.

### 1.2.2 Spontanes Vernetzen

Der Begriff des *spontanes Vernetzen*[1] wird in (Coulouris u. a., 2002) ausführlich diskutiert. Dort heißt es, dass dieser Begriff die Applikationen abdeckt, die Verbindungen von mobilen und nichtmobilen Geräten in Netzwerken unterstützen und zwar auf eine informellere Weise, als es bisher möglich gewesen ist. Im Weiteren besagt Coulouris u. a. (2002), dass die Verbindung zum Netzwerk, sowie der Zugriff auf die Dienste des Netzwerks, möglich sein sollten, ohne dass zuvor eine Benachrichtigung oder administrative Aktionen erforderlich sind. Zusätzlich hierzu werden noch zwei Schlüsselmerkmale des *spontanen Vernetzens*, die einfache Verbindung zu einem lokalen Netzwerk und einfache Integration in lokale Dienste, genannt. Beide Eigenschaften beinhalten die transparente Konfiguration, automatische Verbindung mit Netzwerken und ebenfalls automatisierte Erkennung der im Netzwerk verfügbaren Dienste.

Damit wird unter *spontanes Vernetzen* die Integration von Diensten und Geräten in eine Netzwerkumgebung, mit dem Ziel einer augenblicklichen Verfügbarkeit der Dienste ohne jeglichen manuellen Eingriff, verstanden, wodurch mehr Flexibilität, Mobilität sowie größere Verfügbarkeit erreicht wird.

## 1.3 Nähe Zukunft

Im vergangenen Abschnitt wurde die Lage der Mobilfunktechnologie dargestellt, wie sie heute bereits in verschiedenen Lebensbereichen vorzufinden ist. Diese hatte sich dort erfolgreich etabliert, ist aber noch lange nicht ausgereizt und bietet somit eine Grundlage für weitere, in näherer Zukunft interessante, neue Multimedia Dienste. Diese werden immer weiter Einfluss auf unseres berufliches, sowie privates Leben haben. Mehr noch: Durch diese sich neu eröffnenden Möglichkeiten fließen diese beiden Lebensbereiche immer mehr zusammen. So wird es möglich sein, durch neue Multimedia Dienste, Tätigkeiten aus dem privaten Bereich und von der Arbeit aus ausüben zu können. Auf der anderen Seite werden Multimedia Dienste existieren, die von Zuhause aus, ein schnelles Ausüben der beruflichen Arbeitstätigkeiten erlauben.

In absehbarer Zukunft ist zu erwarten, dass sich so genanntes *elektronisches Geld* stärker verbreitet. In diesem Zusammenhang werden kostenpflichtige Dienste entstehen, die mit dieser Geldform bezahlt werden können. In diesem Bereich werden sehr stark Informationsdienstleistungen vertreten sein. Zusätzlich werden neuen Möglichkeiten der Art und Weise entstehen, wie Bezahlung getätigt werden können. So wird ein Kunde in einem Geschäft, Kino, Restaurant oder Vergleichbaren, die Rechnung nicht explizit durch Abgabe von Banknoten begleichen, vielmehr wird der Betrag von seinem Konto, bei Verlassen des Lokals mithilfe seines mobilen Gerätes, abgebucht. Hierdurch wird in vielen Geschäftsprozessen eine Zeitersparnis entstehen.

[1] Spontaneous Networking

Der Mensch wird immer stärker auf Mobilität getrimmt. Hierdurch entstehende Pausen können in der Zukunft, durch die zur Verfügung stehende mobile Unterhaltungs- oder Lehrdienste, überbrück werden. Vielmehr noch: Reisende werden die Möglichkeit haben, in bestimmten privaten oder geschäftlichen Bereichen, während der Reisen weiter tätig zu sein. Der Mensch wird mit seinem mobilen Gerät im Stande sein, Dienste wie Fernwartung und Support, finanzielle Transaktionen, Überwachung, bis zur Ansteuerung der Zuhause stehenden Kaffeemaschine durchzuführen. In der Zukunft wird den Menschen die Mobilität weniger Zeit rauben.

Stark vertreten werden dann auch Dienste in den sozialen Bereichen vorzufinden sein. So wird es in Bereichen der medizinischen Versorgung möglich sein, entfernte medizinische Ratschläge, wenn nicht sogar Diagnosen, für den Interessenten abgeben zu können. Außerdem kann eine Überwachung der Vitalfunktionen ebenfalls als eine Dienstleistung zur Verfügung gestellt werden. Diese könnte dann von Menschen genutzt werden, bei denen akute Gefahr einer Störung der lebenswichtigen Körperfunktionen besteht, sodass ein Hilferuf an eine nahe Notrufstation gesendet werden kann. Die Helfer können sich während der Hinfahrt auf den Fall durch die Anschauung der Körperfunktionen entsprechend vorbereiten. Durch Ortungsdienste kann dann der Patient schneller gefunden werden.

In der Zukunft wird man wahrscheinlich sehr stark ortsrelevante Dienste vorfinden können. Mit deren Hilfe wird zum Beispiel die Ortung des Patienten, wie das im oberen Szenario geschildert wurde, möglich. Diese könnten aber für weitere Zwecke zukünftig genutzt werden. So werden durch diese Navigationsdienstleistungen (Stau umfahren, nächstes Lebensmittelgeschäft suchen), ortsrelevante Informationsabrufe (Luftverschmutzung, Produktinformation und -suche, Museumsführung), Tracking- und Tracingfunktionen (Paket Zustellung, Kindersuchfunktionen, Ortungsspiele) und vieles mehr möglich sein. Der Kreativität werden hier keinen Grenzen gesetzt. Eines kann man an dieser Stelle sicher festhalten. Die Mobilfunktechnologie und darauf aufbauende Multimedia Dienste, werden in der Zukunft einen großen Einfluss auf unser Leben haben.

## 1.4 Motivation

Im vorangegangenen Abschnitt wurde ein kleiner Überblick über die derzeitige Lage und eine mögliche Entwicklung dieser in der näheren Zukunft gegeben. Es stellt sich nun an dieser Stelle die Frage, was sich die Beteiligten von der Funktechnologie erhoffen. Diese ist nach den Anbietern der Multimedia Dienste und deren Nutzer zu differenzieren. Aus der Sicht eines Betreibers, bietet die neue Sichtweise natürlich ein neues wirtschaftliches Gebiet, auf deren Grundlage neue Dienstleistungen für potenzielle Kunden bereitgestellt werden können, durch welche man sich eine Wertschöpfung erhofft. Es können neue Märkte erschlossen werden. In zweiter Linie kann aber ein Betreiber Dienste zur Verfügung stellen, die er selbst nutzen will. Hierdurch erhofft sich dieser eine Optimierung der Geschäftsprozesse, sodass diese effizienter ausgeführt werden können (Zeit- und Geldersparnis durch verbesserte

Managementmöglichkeiten und breiteren Informationszugriff).

Aus Sicht eines Nutzers bzw. eines Kunden bringen die neuen Dienste ebenfalls eine Möglichkeit, die Lebensweise der Nutzer aus Sicht der Zeitnutzung zu optimieren. Wie auch der Betreiber selbst, kann dieser Dienste auch dazu nutzen, um durch verbesserten Informationszugriff Geldersparnisse, zu erzielen. Neben diesen beiden Zielen wird von dem Verbraucher ein weiterer verfolgt. Dieser wird versuchen sein Lebenskomfort durch vereinfachte und schnelle Abwicklung der Geschäftsprozesse zu erhöhen. In diesem Zusammenhang kann ein vereinfachter ortsunabhängiger Zugriff auf Unterhaltungsmöglichkeiten ebenfalls zu Wohlbefinden des Verbrauchers beitragen. Die Möglichkeit eines einfachen Zugriffes auf verschiedene Ausprägungen der Multimedia Dienste, wie sie in den verschiedenen Bereichen des Lebens auftreten, kann dazu führen, dass der Verbraucher Tätigkeiten ausführt, für welche er vorher nicht die Zeit fand, weil er vielleicht unterwegs war.

Natürlich ergeben sich durch diese neue Sichtweise auch Nachteile. Durch erhöhten Zugriff auf die verschiedenen Dienste, unter anderem der Unterhaltung, besteht die Gefahr, dass ein Kunde (die Gefahr steigt mit abnehmenden Alter) hierdurch nicht nur zu Hause in der freien Zeit, als Beispiel mit Computerspielen beschäftigt ist, sondern sich von diesen noch außerhalb, unterwegs mitreißen läst. Der zuvor erwähnte Vorteil der Zeitersparnis geht hierbei verloren und weitere neue negative Auswirkungen treten an dieser Stelle hervor. Durch die vereinfachte Bezahlungsmöglichkeit kann auch eine Verringerung der Kaufhemmung entstehen, was sich ebenfalls als nachteilig herausstellen kann. Durch das erhöhte Positionstracking kann eine erhöhte, nicht gewollte Kontrolle durch Dritte, entstehen. Dieses kann sich später als eine Art der Freiheitsminderung herausstellen. Weitere Probleme, die ebenfalls nicht vernachlässigt werden dürfen, sind das Thema der Belastung der Umwelt durch Elektrosmog und deren Auswirkungen. Diese wurden jedoch bis heute noch nicht ausreichend erforscht.

Aus technischer Sicht bringt die neue Sichtweise ebenfalls Vor- und Nachteile. So können mithilfe der Ortsunabhängigkeit erstmals bestimmte Geschäftsprozesse ermöglicht werden, die die Mobilität voraussetzten. Auf der anderen Seite bringt die erhöhte mobile Rechenleistung Probleme bei der Energieversorgung. Dafür müssen neuen Energiezellen und sparsame Prozessoren sowie Funkmodule entwickelt werden.

Abschließend kann man sagen, dass die Hoffnungen, die in die neue Sichtweise gesetzt werden, auch Probleme mit sich bringen werden. Diese sollten sorgfältig ausgewogen werden.

## 1.5 Schilderung der Aufgabe

Am Ende dieser Diplomarbeit soll eine Softwarekomponente entstehen. Mit deren Hilfe sollen mobile Geräte im Stande sein, spontan auf Multimedia Dienste in der näheren Umgebung zugreifen zu können. Um das hier verfolgte Ziel etwas näher erläutern zu können, werden kurz zwei Anwendungsszenarien vorgestellt. Das Erste lehnt sich an das allgemeine Vor-

gehen der Benutzung der Multimedia Dienste durch mobile Geräte. Das zweite relevante Szenario ist ein spezialisiertes Hochschulprojekt.

Beginnen wird mit dem Ersten. Dieses knüpft auf die, in den vorangegangenen Unterabschnitten, angesprochenen Anwendungsgebiete an. Man stelle sich vor, man ist gezwungen, eine Geschäftsreise anzutreten. Da diese Aufgabe kurzfristig eingeplant wurde, konnte man sich auf diese nicht vorbereiten, die übrige Zeit reicht auch nicht aus. Auf dem Bahnhof angekommen, stellt man fest, dass der Informationsschalter, bei dem wir nach der schnellsten Verbindung nachfragen wollten, von einer größeren Menschenmenge belagert wird. Ein kurzer Blick zu den Verkaufsschaltern zeigt, dass diese ebenfalls unterbesetzt sind und der Kauf des Tickets wichtige und bereits knappe Zeit verschlingt.

Nachdem wir uns einen kurzen Überblick über die Lage verschafft haben, stellen wir fest, dass dieser Bahnhof mit einem Accesspoint für mobile Geräte ausgestattet ist. Zum Glück haben wir auch unseren PDA dabei. Durch eine kurze Abfrage der Umgebung verbinden wir uns über den aufgefundenen Accesspoint mit dem allgegenwärtigen Netzwerk und bekommen somit eine Liste, der in dieser Umgebung angebotenen Dienste. Darunter finden wir ebenfalls ein von der Bahn zur Verfügung stehenden Multimedia Dienst. Durch entsprechende Eingabemöglichkeiten verbinden wir uns mit diesem. Diese stellt uns eine Möglichkeit dar, bahnrelevante Informationen abzurufen, sowie Kauftransaktionen durchzuführen. Nachdem wir uns mit dem Informationsdienst der Bahn über die Möglichkeiten der schnellsten Verbindung informiert haben, bietet dieser zugleich die Möglichkeit für die gleiche Verbindung einen Fahrkartenkauf zu tätigen. Nachdem wir diesen bestätigt haben, wird von unserem Konto der entsprechende Geldbetrag abgebucht. Durch die Ausführung dieser Transaktion meldet sich ein Multimedia Bankdienst und bietet uns die Möglichkeit, für kleine Transaktionen einen Geldbetrag für die Reise in Form von elektronischem Geld mit auf den PDA zu übertragen. Dieses nehmen wir in Anspruch.

Nachdem wir in dem Zug sitzen, stellen wir fest, dass dieser ebenso mit Accesspoints ausgestattet ist. Durch die entsprechenden Eingaben, zeigt uns unser PDA die Wahlmöglichkeiten, die sich in der näheren Umgebung befinden. Hier finden wir wieder eine ganze Palette der angebotenen Möglichkeiten vor. So finden wir zum Beispiel einen Dienst, welcher eine Verbindung mit dem Internet herstellt. Einige andere bieten uns die Möglichkeit auf verschiedene Unterhaltungsmöglichkeiten zugreifen zu können. So können wir darüber hinaus auf angebotenes Film- oder Rundfunkprogramm zugreifen. Außerdem stehen uns diverse Computerspiele zur Auswahl. Außer den Unterhaltungsdiensten stehen noch andere Dienste zur Verfügung. So kann jederzeit die Information über den Verlauf der Fahrt geholt werden. Über einen weiteren Dienst kann man dem Zugpersonal eine Bestellung über eine mögliche Mahlzeit aufgeben. Das alles ist von jeder Position in dem Zug möglich.

Angekommen in der Zielstadt, suchen wir uns durch einen Positionsdienst den schnellsten Weg aus dem Bahnhof zum nächsten Taxistand. Nachdem wir an der Zieladresse angekommen sind, greifen wir auf den in dem Taxi zur Verfügung stehenden Bezahlungsdienst zu und zahlen den Betrag, ohne sich mit der Kleingeldsuche abgeben zu müssen.

Das Szenario kann beliebig weiter verfeinert werden. Der Fantasie für die Ausprägungen der Multimedia Dienste sind keine Grenzen gesetzt. Durch den in jede Tasche passenden PDA sind wir im Stande, viele der geschäftlichen und privaten Transaktionen von überall zu tätigen. Das Tragen eines großen, unhandlichen Notebooks und das Suchen einer geeigneten Netzwerkdose wird nicht mehr notwendig sein. Das Notwendige hierzu finden wir um uns herum. Das Netzwerk der Multimedia Dienste ist allgegenwärtig.

Das eben vorgestellte Szenario zeigt die Einsatzmöglichkeiten verschiedener Softwarekomponenten, wenn diese durch entsprechende Erweiterungen in die Infrastruktur für Multimedia Dienste integriert werden. Dieses war eher theoretischer Natur und sollte deswegen anhand eines konkreten Szenarios veranschaulicht werden. Im Folgenden wird ein Szenario vorgestellt, welches tatsächlich an der HAW-Hamburg durchgeführt werden soll, und somit die Zielanforderung für die hier entwickelte Softwarekomponente darstellt.

Im Rahmen der Robotliga soll eine Plattform für das Roboterfußball entwickelt werden. Die Regeln sind einfach: Zwei Roboter spielen gegen zwei andere Roboter. Beide Mannschaften versuchen, den Ball in dem Tor der gegnerischen Mannschaft zu plazieren.

So einfach wie es klingt, ist dieses aus der technischen Sicht sehr aufwendig. Die Roboter nehmen ihre Umgebung mit Hilfe von Sensoren wahr, wie Fühler und Sonar. Diese erlauben eine eingeschränkte Reaktionsmöglichkeit auf die Umgebung, und somit auch einfache Spielstrategien zwischen den beiden Mitspielern untereinander. Aus diesem Grund wird zusätzlich zu den Standardsensoren, noch ein anderer Sensor den Robotern zur Verfügung gestellt. Über dem Spielfeld wird eine Kamera installiert, welche dem Roboter deren Position übermittelt. Dieses wird mit Hilfe von speziellen Markierungen, welche auf den Robotern platziert sind, berechnet. Die Berechnung der Position wird durch eine, im Rahmen einer Diplomarbeit von Revout (2003) realisierten Anwendung durchgeführt. Dieser rechenintensive Algorithmus kann im Vergleich zu der Auswertung der Standardsensoren, nicht von dem Roboter selbst berechnet werden. Aus diesem Grund wird hierzu ein Server zur Verfügung gestellt. Dieser (Positions-) Server bekommt von einem anderen (Kamera-) Server die Daten der installierten Kamera, durch welche die Position der Roboter und des Fußballs berechnet werden kann.

Wie man sich denken kann, sind die Roboter ständig in Bewegung, so wie im richtigen Fußball. Man kann sich leicht vorstellen, dass in dem ganzen Zusammenspiel der Roboter, diese nicht über ein Netzwerkkabel mit dem (Positions-) Server, sowie miteinander verbunden sein können. Es steht außer Frage, dass diese über kabellose Kommunikation miteinander und mit dem Server kommunizieren müssen.

An dieser Stelle wird die, in dieser Arbeit entwickelte, Softwarekomponente eingesetzt, welche den Robotern erlaubt, auf Dienste des (Position-) Servers, sowie alle anderen verfügbaren Dienste spontan im mobilen Zustand zuzugreifen. Diese Softwarekomponente nimmt den Robotern die gesamte, mit der Suche, Kommunikation und Koordination anfallende Arbeit ab. Diese können sich vollständig auf die erforderliche Datenkommunikation einstellen.

## 1.6 Übersicht über die Arbeit

In dieser Arbeit wird wie folgt vorgegangen. In dem Kapitel 2 wird als Erstes die hier zu bearbeitende Aufgabe genau beschrieben und die Zielsetzung spezifiziert. Dazu werden die zuvor ermittelten Anwendungsszenarien genauer nach den, in diesem Zusammenhang entstehenden, Problemen hin untersucht. Dabei wird die Zielsetzung aus der Sicht des Benutzers und Informatiksicht durchgeführt. Nachdem die Probleme erkannt wurden, wird die Materie der *verteilten Systeme* ausreichend verdeutlicht, da dieses Wissen für das Verständnis der entstehenden Probleme obligatorisch ist. Dabei werden die gängigen Technologien genannt, wobei die, die für diese Arbeit relevant sind, näher beschrieben werden. Als Nächstes werden dann die Anforderungen betrachtet, die durch mobile Geräte an eine Software gestellt werden. Zum Schluss werden die verschiedenen Ansätze auf ihre Vor- und Nachteile gegeneinander verglichen. Daraus hervor gehende Technologien werden dann in dem Lösungsansatz verwendet.

Im Kapitel 3 wird der Entwurf des hier zu realisierenden Softwaresystems besprochen. Als Erstes werden die für dieses System spezifischen Randbedingungen festgelegt. Um das weitere Vorgehen zu vereinfachen, werden als Nächstes die konzeptionellen Entscheidungen getroffen. Danach wird zu dem eigentlichen Softwareentwurf übergegangen. Dabei werden die im Kapitel 2 identifizierten Anwendungsfälle weiter verfeinert, sodass sich einige Systemkomponenten daraus herauskristallisieren. Anhand dieser wird eine grobe Systemarchitektur entworfen. Im Weiteren werden die Systemkomponenten weiter granuliert und in den modularen Aufbau des Systems integriert. Danach wird der Verlauf der Dienstkommunikation festgelegt. Zum Schluss werden Testfälle vorgestellt, die die Funktionsfähigkeit des Systems bestätigen sollen.

Nachdem die Systemarchitektur und darin enthaltene Komponenten festgelegt wurden, wird im Kapitel 4 der Entwurf umgesetzt. Dazu wird zunächst die für die Realisierung notwendige Laborumgebung beschrieben. Nach der Festlegung der Laborumgebung wird auf die Einzelheiten der Umsetzung des Konzepts eingegangen. Dabei werden die hierbei verwendeten Muster genannt. Zum Schluss wird die Umsetzung des Kommunikationsprotokolls beschrieben.

In dem letzten Kapitel 5 werden die im Rahmen dieser Arbeit gesammelten Erfahrungen, Erkenntnisse, Kritiken und Zukunftsvisionen besprochen. Dabei wird zuerst die gesamte Arbeit zusammengefasst. Anschließend wird die Arbeit bewertet, indem der Entwicklungsstand beschrieben wird, gewonnene Erkenntnisse genannt werden und anschließend Kritik ausgeübt wird. Zum Schluss wird ein Ausblick gegeben wie diese Arbeit weiter fortgesetzt werden kann.

# Kapitel 2

# Analyse

## 2.1 Einleitung

Die zuvor genannte Aufgabenstellung weist viele Hürden auf, welche auf dem Weg entstehen, das allgegenwärtige Netzwerk für mobile Benutzer bzw. Geräte zugreifbar zu machen. Mehr noch, die allgegenwärtige Infrastruktur sollte selbst eine höhere Abstraktion aufweisen, sodass an dieser Stelle nicht mehr von Methodenaufrufen, sondern Dienstleistungen gesprochen wird.

Dieses Kapitel beschäftigt sich mit dem Problem der Abstraktion und anderen Problemstellungen, indem es als Erstes die Problemstellung in dem Abschnitt 2.2 näher beschreibt, sodass eine ausreichende Einführung in die Problemstellung gegeben wird. Hierzu werden drei Anwendungsszenarien zur Hilfe genommen. Anhand derer wird verdeutlicht, auf welche Probleme ein möglicher Lösungsansatz stoßen wird. Diese Probleme werden dabei aus zwei Sichten identifiziert, der Benutzer- und Informatiksicht.

Nachdem die Probleme erkannt wurden, wird in Abschnitt 2.3 das heute verfügbare Wissen nach bereits vorhandenen Lösungsansätzen abgesucht. Um ein höheres Verständnis der Problematik zu erreichen, wird zu Beginn des Abschnittes das Gebiet der „Verteilten Systeme“ erläutert. Dabei wird versucht, die Positionierung der Problemstellungen durchzuführen. Im Anschluss werden dann einige der heute bekannten Lösungsansätze vorgestellt.

Als Nächstes werden in dem Abschnitt 2.4.3 die zuvor betrachteten Lösungsansätze auf ihre Eignung in der Realisierung analysiert. Hierzu werden zuerst Anforderungen identifiziert, die bei der Bewertung berücksichtigt werden sollen. Die Anforderungen selbst finden ihren Ursprung in dem Abschnitt 2.2, wobei diese an dieser Stelle konkretisiert werden sollen. Bei der Veranschaulichung der einzelnen Anforderungen, werden die verwandten Lösungsansätze aus dem Abschnitt 2.3 aufgegriffen, um die erkannten Anforderungen festigen zu können.

Im abschließenden Abschnitt wird eine Diskussion der bereits existierenden Lösungsansätze durchgeführt. Dabei werden die zuvor erkannten Anforderungen mitberücksichtigt, sodass als Ergebnis der Diskussion eine Auswahl der erforderlichen Ansätze getroffen werden kann, die zur Erreichung des gestellten Ziels notwendig sind.

## 2.2 Was ist das Problem

### 2.2.1 Szenarien

Einer der wesentlichen Aufgaben einer wissenschaftlichen Arbeit, ist die eindeutige Benennung der Probleme, die gelöst werden sollten. Hierdurch wird das Problemgebiet besser verinnerlicht und die eigentliche Problembearbeitung begünstigt. Hierzu werden einige Anwendungsszenarien zu Hilfe genommen und beschrieben.

#### Bahn Information

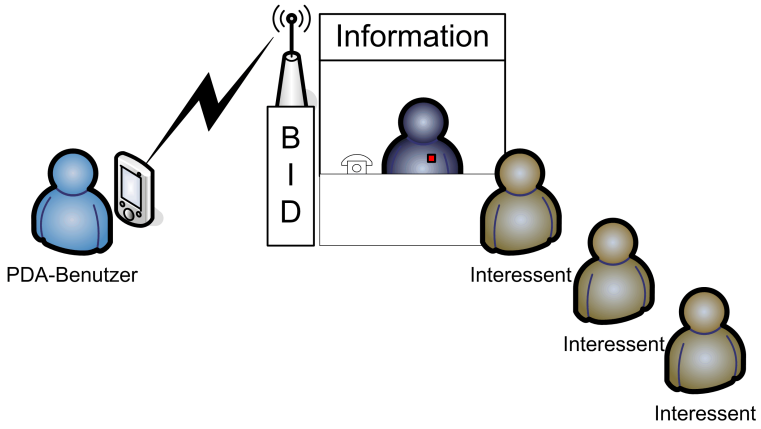


Abbildung 2.1: Bahn Informationsszenario.

In diesem Anwendungsszenario spielt die Informationsbeschaffung die wesentliche Rolle. Es handelt sich um eine Metapher für einen Bahninformationsschalter. Man stelle sich vor, alle auf dem Bahnhof vorhandenen Informationsschalter sind nicht besetzt oder durch große Menschenmengen überfordert. Jetzt stellen sie sich vor, dass sie an diesem Bahnhof soeben angekommen sind. Sie wissen, dass ihr Zug innerhalb von einigen Minuten abfahren sollte, sie haben aber bei dem ganzen Durcheinander die genauere Abfahrtszeit und den Bahnsteig vergessen. Es besteht auch keine andere Auskunftmöglichkeit. Durch einen kurzen Blick auf die belagerten Informationsschalter, fällt ihnen ein Informationssymbol auf, welches auf die Präsenz von Multimedia Diensten in diesem Bereich hinweist. Zum Glück haben sie ihren PDA dabei. Dieser ist mit einem Funkmodul versehen, wodurch ein Zugriff auf das Funknetzwerk in der näheren Umgebung möglich ist. Durch kurzes Abtasten der Umgebung identifizieren sie mehrere verfügbare Dienste. Einer darunter ist ein Bahninformationsdienst (BID).

Sie wählen diesen aus und verbinden sich mit ihm. Kurz darauf erscheint eine Auswahlmöglichkeit der Informationen, die sie abrufen können. Darunter finden sie auch die gesuchte

Fahrplanauskunft. Durch entsprechende Eingaben wird nach kurzer Zeit die Information über die Abfahrtzeit und Bahnsteig auf Ihren PDA übermittelt.

**Roboter-Fußball**

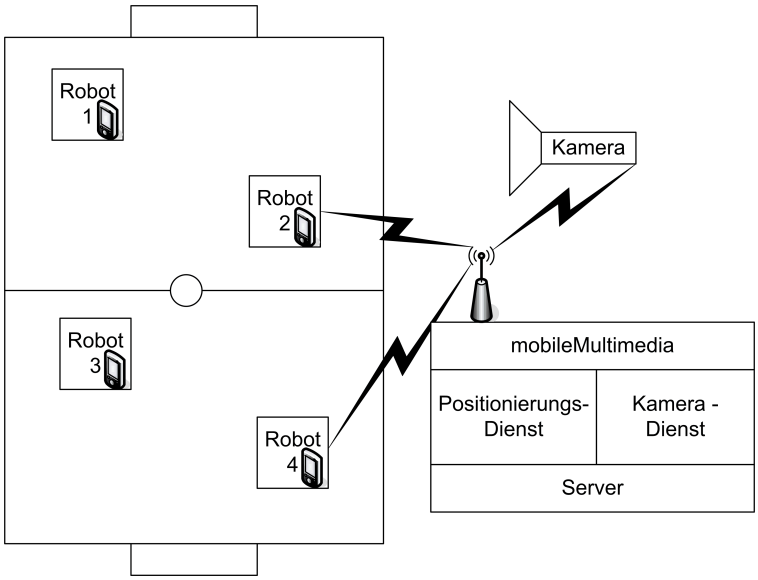


Abbildung 2.2: Roboterfussballszenario.

Das zweite Anwendungsszenario ist eine Spezialisierung des oben beschriebenen Szenarios. Da dieses aber eine Einsatzmöglichkeit beschreibt wie diese Software später in der HAW-Hamburg eingesetzt werden soll, wird dieses an dieser Stelle ebenfalls beschrieben. Es soll eine Plattform für mobile Roboter geschaffen werden, auf der diese in einem Mannschaftsspiel, in diesem Fall Fußball, gegeneinander spielen.

Diese Plattform besteht aus einem Spielfeld, auf dem zwei Mannschaften, bestehend aus zwei Robotern, den Ball in das gegnerische Tor zu platzieren versuchen. Der Ball ist durch Infrarotsender für die Roboter sichtbar. Das Spielfeld und die Roboter sind mit besonderen Markierungen versehen. Zusätzlich zeichnet eine Kamera das ganze Spielgeschehen auf. Diese aufgezeichneten Daten werden zur Echtzeit zu einem Kameraserver übertragen, wo diese dann durch einen entsprechenden Algorithmus ausgewertet werden. Dieser Algorithmus ist der Gegenstand einer Diplomarbeit von Ilia Revout (Revout, 2003) und hat als Aufgabe, mithilfe der Markierungen die genaue Position der Roboter und deren Ausrichtungswinkel zu bestimmen. Während des Spielverlaufs sollten die Roboter dann auf die Ergebnisse der Positionsberechnung zugreifen und damit das weitere Spielgeschehen planen können. So suchen diese die Umgebung nach dem entsprechenden Positionierungsdienst ab und verbinden sich mit ihm, um die benötigten Daten abzurufen. In der Abbildung 2.2 ist das eben geschilderte Szenario kurz skizziert.

**Ferienclub Dienstleistungen**

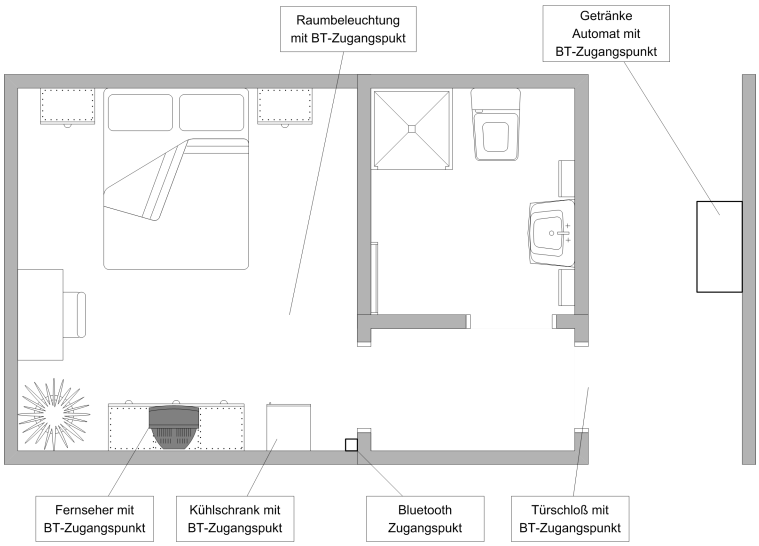


Abbildung 2.3: Multimedia-Ferienclubzimmer.

Um die Anwendungsmöglichkeiten noch mehr zu verdeutlichen, wird an diese Stelle ein weiteres Szenario als Ergänzung zu Hilfe genommen. Dieses wird wie das erste nicht im Rahmen dieser Arbeit implementiert, ist aber mithilfe der hier implementierten Software möglich. Dieses Szenario beschreibt die Möglichkeiten, die sich in einem Ferienclubgebäude durch die Multimedia Dienste ergeben.

So werden Räumlichkeiten des Ferienclubs an Gäste verbucht. Des Weiteren werden weitere Dienste zur Verfügung gestellt, die von den Gästen genutzt werden können. Die hier zu entwickelnde Software kann auch in diesem Umfeld neue Möglichkeiten eröffnen. So kann bei der Ankunft in dem Ferienclub, bei der Rezeption, nach Beendigung aller Formalitäten, an den Gast ein PDA ausgehändigt werden. Dieser ist mit Zugangssoftware ausgestattet, die Zugriff auf die, in dem Ferienclub zur Verfügung gestellte Dienste ermöglicht. Außerdem ist der PDA mit Informationen wie Zugangsberechtigung des Gastes geladen. So stellt der PDA in diesem Fall eine Informations-, Zugriffs- und Bezahlungsplattform dar.

Unter der Informationsplattform wird Folgendes verstanden: Bekommt der Gast den PDA in die Hand, kann dieser über den Positionierungsdienst die Information über seine jetzigen Position und den Weg zu seinem Ferienclubzimmer abrufen.

Dieser kann ebenfalls sofort das Unterhaltungsprogramm, welches der Ferienclub anbietet, für den Abend abrufen. Daneben kann dieser ein Interessenprofil anlegen, wodurch der Ferienclub ihn nur über für ihn interessante Ereignisse informieren wird. Der PDA ist somit eine Informationsplattform zwischen dem Ferienclub und dem Gast.

Die Zugangsplattform stellt Möglichkeiten bereit, durch die ein Zugriff auf die im Ferienclub befindlichen Dienste ermöglicht wird. Es kann eine ganze Reihe von Diensten angebo-

ten werden, nicht alle sind aber für den Gast sichtbar, da diese zum Beispiel nicht in dem gebuchten Leistungsumfang enthalten waren. So kann man sich vorstellen, dass der Gast die Möglichkeit des Internetanschlusses nicht in seinem Leistungspaket dabei hätte. Der Zugang auf das Internet würde ihm hierdurch nicht zur Verfügung stehen. Die Zugangsplattform regelt dieses Problem.

Die Bezahlungsplattform, wie der Name bereits sagt, bietet die Möglichkeiten einer expliziten Bezahlung. So können Dienste im Ferienclub installiert sein, die zu keinem Leistungspaket dazugehören, trotzdem angeboten werden können (Spielkasino, Getränkeautomat). Hier können sichere Geldtransaktionen von dem Gast durchgeführt werden. Die Abbildung 2.3 zeigt einige Einsatzmöglichkeiten für die Middleware. Dieses Szenario ist der Gegenstand der Diplomarbeit von André Lüpke (Lüpke, 2004) und wird dort näher beschrieben.

### 2.2.2 Probleme Identifizieren

Die in dem oberen Abschnitt 2.2.1 beschriebenen Anwendungsszenarien stoßen bei deren Umsetzung auf einige Probleme, die eine nähere Betrachtung erfordern. Hierzu werden diese Szenarien auf Probleme abgesucht und aufgelistet.

**Verbindungsaufbau** Als Hauptproblem bei allen drei Szenarien ist wohl die Problematik des Verbindungsaufbaus zu nennen. Diese wird besonders durch den ständigen Wechsel der Umgebung zusätzlich erschwert, da sich das Gerät von Zugangspunkt zu Zugangspunkt durchhangeln muss.

**Unterbrechung** Ein weiteres Problem stellt die Aufrechterhaltung der Verbindung zu einem Zugangspunkt dar. Dieses Problem ist durch die verschiedenen Störquellen sehr schwierig im dem Griff zu bekommen. Eine Verbindungsunterbrechung kann z.B.: durch die eingeschränkte Reichweite bzw. Störung der Funkübertragung und Auslastung des Zugangspunktes verursacht werden. Dabei ist noch zu unterscheiden, dass die Verbindungsunterbrechung auch eine beabsichtigte Maßnahme sein kann, zumindest auf der physikalischen Ebene. Diese tritt bei dem Wechsel des Zugangspunktes auf und gilt für die Unterstützung der Mobilität als Voraussetzung.

**Wiederherstellung** In Zusammenhang mit Verbindungsunterbrechung taucht das Problem des Wiederaufbaus bzw. Wiederherstellung der Verbindung auf. Dabei ist unter Wiederaufbau, der Aufbau der Verbindung zu dem Zugangspunkt gemeint. Bei der Wiederherstellung ist die Wiederherstellung des Zustandes und der Daten, so wie sie kurz vor der Verbindungsunterbrechung waren, zu verstehen. So soll der Benutzer bei einer ungewollten Verbindungsunterbrechung bei seiner bereits angefangenen Aufgabe fortfahren und nicht wieder von vorne anfangen. Dieses Problemszenario tritt bei allen drei Szenarien auf.

**Spontanität** Weniger bei dem Roboter-Szenario, mehr bei den anderen Szenarien, tritt in Zusammenhang mit der Problematik der Netzwerkkommunikation, die Eigenschaft der Spontanität der Kommunikation im Vordergrund. Die Spontanität wird hier vor allem bei den oberen drei Eigenschaften vorausgesetzt, um somit ein Fundament für weitere wichtigere Eigenschaften zu schaffen.

**Netzwerktransparenz** Ein weiterer Punkt ist die Netzwerktransparenz. In erster Linie sollen die Benutzer und die Entwickler der Multimedia Dienste von der Verteilung des Systems sowie der Netzwerkinfrastruktur nicht betroffen sein. Das Problem der Konfiguration des Netzwerkes und der Verteilung des Systems soll entfallen.

**Benutzer Identifizierung** In allen Szenarien wird eine Authentifizierung des Benutzers erforderlich[1]. Bei der Fülle von Benutzern müssen diese jeweils identifiziert werden, um weitere wichtige Aspekte ermöglichen zu können.

**Sicherheit** Bei der Übertragung werden viele Daten übertragen. Diese sind verschiedener Natur. Manche sind einfache anwendungsspezifische Daten, bei dem die Sicherheit keine Rolle spielt. Es werden aber auch Daten versendet, die kritisch sind und für Dritte nicht lesbar sein sollen. An dieser Stelle muss entsprechend ein Sicherheitskonzept zur Verfügung gestellt werden.

**Darstellung** Ebenfalls wichtig für die Szenarien, in denen der Mensch direkt mit den Diensten interagiert, ist eine vernünftige Darstellung der Dienste, um die angeforderten Informationen lesbar für diesen darstellen zu können. Dabei tritt ein zusätzliches Problem der Darstellung der Informationen auf, die auf das zugreifende Gerät angepasst sein sollte und somit auf seine Darstellungsstärken oder Schwächen zugeschnitten sind.

**Performance** Ein weiteres Problem stellen die Geräte bzw. die Rechenleistung, die diese anbieten, dar. Speziell in dem Roboter-Szenario ist diese sehr wichtig, da der Roboter Informationen in Echtzeit anfordert, damit dieser auf die Umgebung reagieren kann. Eine höhere Bewegungsgeschwindigkeit und damit verbundener schneller Positionswechsel sollte ebenfalls berücksichtigt werden.

**Vielfältigkeit** Bei der Fülle, der im Ferienclub oder auf dem Bahnhof verfügbaren Diensten kann deren Vielfältigkeit zu einigen Kompatibilitätsproblemen führen.

**Administration** Die Administration einer großen Ansammlung von Diensten, stellt ebenfalls eine große Hürde dar. Diese müssen wartbar sein.

**Verteilung** Wie soll die Anwendung verteilt werden, damit das mobile Gerät nicht zu sehr mit dieser belastet wird und somit anderen Aufgaben nicht mehr nachgehen kann?

[1] Jeder Robot erhält nur die für ihn gemeinten Informationen.

**Skalierbar** Außer dem Roboter-Szenario muss in beiden anderen Szenarien eine Skalierung der Architektur möglich sein. Es ist leicht nachvollziehbar, dass in beiden Fällen eine unterschiedliche Nutzungsbreite bereitgestellt werden soll. An einen kleinen Bahnhof wird sich vielleicht ein Benutzer pro Minute anmelden. Auf einem Hauptbahnhof werden das vielleicht 10 oder mehr sein. In beiden Fällen sollte das System immer noch akzeptabel arbeiten können.

Die hier aufgelisteten Probleme unterscheiden sich in Ihrer Natur und Wichtigkeit. Natürlich werden Neue entstehen, die erst in der näheren Betrachtung bzw. in der Testphase zum Vorschein treten. Diese Liste hat als Aufgabe die Fülle der Probleme, die hierbei entstehen, zu verdeutlichen und weniger alle aufzulisten. In den folgenden beiden Abschnitten sollen die Probleme aus zwei Sichten noch einmal erläutert werden, um somit den Problembereich einzugrenzen.

Diese Betrachtung erfolgt aus der Benutzer- sowie der Informatiksicht und trennt diese voneinander. Die Benutzersicht beinhaltet die benutzerrelevanten Anforderungen. Die Informatiksicht dagegen beinhaltet die Inhalte der Informatik, die für den Benutzer nicht direkt sichtbar, bzw. für diesen vollständig transparent sind.

### Benutzersicht

**Verbindungsaufbau** Der Verbindungsaufbau stellt für den Benutzer nur im Fall der physikalischen Verbindung ein Problem dar. Dieser wird durch die erforderlichen Netzwerkeinstellungen und Auswahl des Zugangspunktes überfordert. Der physikalische Verbindungsaufbau stellt nur eine Notwendigkeit dar, um auf entsprechende Multimedia Dienste zugreifen zu können und hält den Benutzer von der eigentlichen Aufgabe ab.

**Unterbrechung** Das Gleiche betrifft die Verbindungsunterbrechung. Diese stellt für den Benutzer sofort einen Fehlerzustand dar, welcher ihn daran hindert, die angefangene Aufgabe zum Abschluss zu bringen.

**Wiederherstellung** Dieser Punkt schließt die Problematik des Verbindungsaufbaus mit ein und erweitert ihn um die Probleme, auf die der Benutzer stößt, wenn dieser den vorherigen Zustand einer abgebrochenen Verbindung wieder herstellen muss. Dieses kann man dem Benutzer ebenfalls nicht zumuten.

**Darstellung** Die einheitliche Darstellung der Multimedia Dienste stellt eine große Herausforderung dar. Der Benutzer will von den angebotenen Diensten Gebrauch machen, ohne sich dabei auf ein mobiles Gerät beschränken zu müssen.

**Performance** Geht eine langsame Anwendung den gestellten Aufgaben nicht oder nur mühsam nach, ist diese für den Benutzer von nicht all zu großem Interesse. Die Bearbei-

tung und somit die Antwortzeit sollte sich im, für den Benutzer zumutbarem, Rahmen bewegen.

**Vielfältigkeit** Bei der großen Menge der angebotenen Multimedia Dienste, ist schnell der Überblick verloren. Die Vielfältigkeit sollte für den Benutzer übersichtlich gemacht werden können.

**Administration** Aus Sicht einer spezialisierten Benutzermenge, wie das bei den Administratoren der Fall ist, ist durch die Vielfalt der Multimedia Dienste, deren Administration erschwert.

**Leicht bedienbar** Die Anwendung soll leicht bedienbar sein. Der Benutzer soll eine Anwendung vorfinden, von der er schnell und gezielt Gebrauch machen kann.

#### Informatiksicht

**Verbindungsaufbau** Eine große Herausforderung stellt der Aufbau einer physikalischen Verbindung mit dem Netzwerk dar, wobei hierdurch kein negativer Einfluss auf die Mobilität ausgeübt werden darf. Zusätzlich hierzu entsteht ein Problem der Auffindung solcher physikalischen Zugangspunkte. Wie werden diese auf ihre Verbindungsqualität bewertet? Es sollen keine Verbindungen aufgebaut werden, bei dem sofort mit einer Unterbrechung zu rechnen ist.

**Unterbrechung** Wenn die Verbindung unterbrochen wird, wie soll das System darauf reagieren? Von vorne kann man zwei Arten der Unterbrechung identifizieren. Die Eine wird durch die Mobilität selbst verursacht, ist also gewollt. Die Andere, Ungewollte kann aus vielen Gründen verursacht werden. Das System muss zwischen den beiden unterscheiden können und darauf entsprechend reagieren.

**Wiederherstellung** Beim Wiederherstellen einer physikalischen und logischen Verbindung muss der Zustand dieser reproduzierbar sein. In diesem Zusammenhang muss beantwortet werden, wie und wo der Zustand der Verbindung persistent gemacht wird.

**Netzwerk** Die mobilen Geräte wechseln ihre physikalische und die logische Umgebung ständig. Dabei wird ebenfalls der Zugriffspunkt gewechselt. Es stellt sich die Frage, wie man den Benutzer beim Wechsel trotzdem als den Gleichen erkennt und seine Aktion als Wechsel eines Zugangspunktes deutet und nicht als gewollten Abbau der Verbindung.

**Benutzer Identifizierung** Ein weiteres großes Problem, welches im Zusammenhang mit offener und spontaner Umgebung auftaucht, ist die Identifizierung und Authentifizierung des Benutzers.

**Sicherheit** Ein offenes System stellt ebenfalls viele neue Möglichkeiten für unbefugte Zugriffe und die damit verbundenen Risiken offen.

**Darstellung** Wie können Multimedia Dienste konzipiert werden, damit diese eine Darstellung anbieten können, ohne zu wissen, auf welchem Gerät diese dargestellt werden sollen?

## 2.3 Verteilte Systeme

Nachdem im vorherigen Abschnitt die entstehenden Probleme dargestellt wurden, soll an dieser Stelle ein Einblick in die heute bereits vorhandenen und bewährten Technologien gegeben werden. In Umfeld der beschriebenen Szenarien könnten diese Technologien dabei helfen, entstehende Probleme zu beseitigen. Neben den Technologien selbst, wird die für das Verständnis notwendige Theorie über *Verteilte Systeme* gegeben, da diese als Grundlage für den Entwurf und Realisierung der Anwendungsszenarien zu sehen ist.

### 2.3.1 Allgemein

Bei der Analyse der hier betrachteten Problemstellung, ist es wichtig, etwas Allgemeinwissen über das Gebiet der verteilten Systeme zu haben. So soll hier ein Einblick in die allgemeinen Ansätze gegeben werden, die in diesem Bereich sich bis heute bewährt haben. Zusätzlich wird versucht, den hier betrachteten Fall zu identifizieren und allgemein abzubilden. Im Zusammenhang mit den verteilten Systemen werden die hier verwendeten Technologien grob erklärt.

#### Verteilte Systeme - Definition

In Tanenbaum und van Steen (2002) wird folgende Definition für ein verteiltes System gegeben:

> A distributed system is a collection of independent computers that appears to its users as a single coherent system.

Mit anderen Worten kann ein Verbund von Computern (ohne deren Rechenkapazität und Größe zu betrachten) und Softwarekomponenten (die Komplexität spielt ebenfalls keine Rolle), bei dem diese miteinander vernetzt sind (die Verbindungsart ist ebenfalls zu abstrahieren) und zwischen denen auf der Basis von Nachrichten eine Kommunikation und Koordination zustande kommt, als verteiltes System definiert werden. Dabei spielt es ebenfalls keine Rolle, wie weit diese verteilten Komponenten auseinander liegen. So ist jede Ausprägung der Distanz zwischen denen möglich.

**Middleware - Definition**

In Coulouris u. a. (2002) wird der Begriff „Middleware“ als eine Softwareschicht identifiziert, die eine Programmierabstraktion bereitstellt und die Heterogenität der zu Gründe liegenden Netzwerke, Hardware, Betriebssysteme und Programmiersprachen verbirgt.

Außerdem wird in Tanenbaum und van Steen (2002) „Middleware“ als eine Softwareschicht identifiziert, die aufgesetzt auf das Netzwerkbetriebssystem mehr oder weniger die Heterogenität der Ansammlung darunter liegender Plattformen nach außen verbirgt, gleichzeitig die Vernetzung dieser unterstützt und die Verteilung dieser transparent macht. Daneben darf diese die Sicht und die Souveränität eines allein stehenden Rechners nicht stören.

Middleware wird in beiden Fällen als eine Hilfssoftwareschicht erkannt, welche die Aufgabe hat, Probleme der Heterogenität der unterschiedlichsten Plattformen und Kommunikationsprotokolle zu lösen. Zusätzlich stellt die Middleware mithilfe der Verteilungstransparenz ein einheitliches Programmiermodel sowie Middleware Dienste bereit, die von den verteilten Anwendungen genutzt werden.

In den verschiedenen Ausprägungen der Middleware wird diese jeweils abhängig von der Abstraktion, durch Prozesse oder Objekte, in mehreren Rechnern dargestellt. Diese arbeiten zusammen, um die Kommunikation und die gemeinsame Nutzung von Ressourcen für verteilte Anwendungen zu unterstützen. So ist die Aufgabe der Middleware, ein solides Fundament aus praktischen Diensten bereitzustellen, wodurch der Aufbau von verteilten Softwarekomponenten begünstigt wird und eine Zusammenarbeit untereinander stattfinden kann. Weiterhin unterstützt die Middleware Abstraktionen, wodurch die Kommunikationslogik von der Anwendungslogik gehoben wird.

Für die Realisierung eines verteilten Systems kann die Middleware auch Dienste für verteilte Softwarekomponenten bereitstellen. Es handelt sich dabei um infrastrukturelle Dienste. Diese hängen stark mit dem von der Middleware verfolgten Programmiermodel zusammen. Außerdem stellen einige Ausprägungen der Middleware Möglichkeiten dar, zusätzlich notwendige Dienste für die verteilten Anwendungen zu entwickeln.

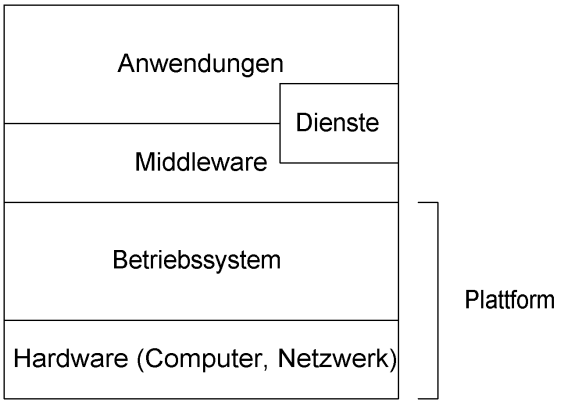


Abbildung 2.4: Positionierung der Middleware

In verteilten Anwendungen müssen die beiden zu kommunizierenden Seiten ihre Positionen gegenseitig nicht kennen. Middleware stellt Ortstransparenz zur Verfügung. Weiterhin stellt diese die Kommunikationsprotokolle bereit, welche unabhängig von den zugrunde liegenden Transportprotokollen sind. Die Differenzen, die auf Grund der unterschiedlichen Hardwarearchitekturen entstehen, werden durch geeignete Marshalling und Unmarshalling Methoden ebenfalls durch die Middleware verborgen. Die Betriebssystemschicht wird mithilfe der Middleware ebenfalls abstrahiert. Bei manchen Middlewareabstraktionen werden außerdem die in der verteilten Anwendung zum Einsatz kommenden Programmiersprachen durch Middlewareschicht abstrahiert. Abbildung 2.4 zeigt die Positionierung der Middleware im Systemkontext.

Gewöhnlich wird die Middleware in einer schichtenorientierten Architektur zwischen einer höheren Anwendungs- bzw. Benutzerschicht und der untersten Betriebssystemschicht positioniert. In der Abbildung 2.5 wird die Positionierung der Middleware in einem Netzwerkkontext dargestellt.

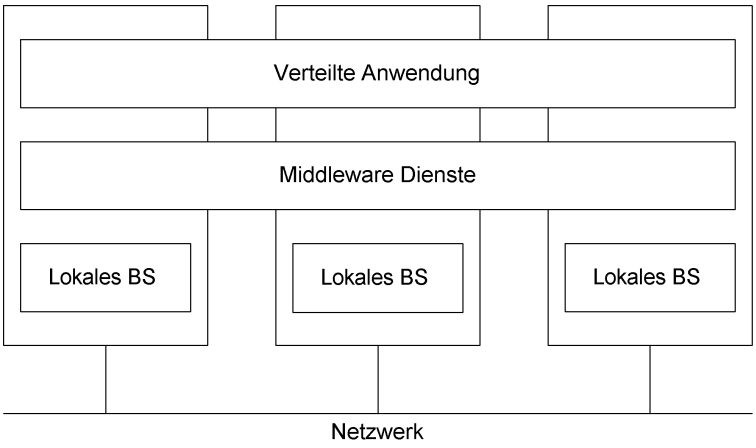


Abbildung 2.5: Grobe Struktur eines verteilten Systems (Tanenbaum und van Steen, 2002).

### Eigeschaften eines Verteilten Systems

Woran kann man erkennen, dass ein System ein verteiltes System ist? Entsprechend der von Tanenbaum und van Steen (2002) aufgestellter Definition, besteht ein verteiltes System aus mehreren autonomen Rechnern, die eigenständig funktionieren. Zusätzlich besitzt ein verteiltes System sechs charakteristische Merkmale, wodurch dieser sinnvoll genutzt werden kann.

**Ressourcenteilung** So ist unter dem Merkmal Ressourcenteilung die Nutzung der gemeinsamen Ressource des verteilten Systems wie Hardware, Software und gemeinsamer Daten zu verstehen. Dieses Merkmal stellt auch die Hauptmotivation für den Aufbau

von verteilten Systemen dar. Hierdurch werden die verteilten Ressourcen effizienter genutzt.

**Offenheit** Ein weiteres wichtiges Merkmal ist die Offenheit. Verteilte Systeme sollten erweitert werden können. Dieses geschieht, indem neue Dienste oder gemeinsam nutzbare Ressourcen eingefügt werden. Dabei dürfen die bereits bestehenden Dienste nicht beeinträchtigt werden. Durch die Offenheit, die durch das Offenlegen der Spezifikation und Dokumentation der wichtigsten Softwareschnittstellen erreicht wird, ist eine Erweiterung des verteilten Systems begünstigt.

**Gleichzeitigkeit** Gleichzeitigkeit bedeutet, dass mehrere Prozesse zur gleichen Zeit ausgeführt werden können, ohne sich gegenseitig zu beeinflussen. So muss das verteilte System sicherstellen, dass bei mehreren gleichzeitigen Zugriffen auf eine Ressource, diese darauf ausgelegt ist und somit in einer nebenläufigen Umgebung sicher ist.

**Skalierbarkeit** Mit Skalierbarkeit ist die Erweiterbarkeit eines verteilten Systems um neue Hardware gemeint, wobei das ursprüngliche System hierbei sein Verhalten nicht ändert. Hierdurch wird eine Verbesserung der Performance des Systems erreicht.

**Fehlertoleranz** Im verteilten System können viele verschiedene Fehler auftreten. So soll das System bei möglichen Soft- bzw. Hardwarefehlern diese entsprechend verarbeiten können und weiter verfügbar bleiben. Dieses Merkmal wird als Fehlertoleranz bezeichnet.

**Transparenz** Das sechste und somit letzte Merkmal ist die Transparenz. Mithilfe der Transparenz wird das Ziel verfolgt, die Verteilung der Komponenten des verteilten Systems vor dem Anwender zu verbergen. So sind für diesen die Position und Details der Bearbeitung nicht relevant. Der Anwender nimmt das verteilte System als eine Einheit wahr.

Als Beispiel für ein verteiltes System kann das Internet genannt werden, welches die oben genannten Merkmale in unterschiedlichem Grad aufweist. Dieses stellt die verschiedenen Dienste bereit, wie das World Wide Web, E-Mails oder Datenübertragung und das von verschiedenen, in der ganzen Welt verteilten, Orten aus. Dieses wächst jeden Tag weiter und unterstützt immer mehr und immer unterschiedlicher werdende Dienste.

Als ein zweites Beispiel eines verteilten Systems und somit auf die Aufgabenstellung, die in dieser Arbeit bearbeitet werden soll, Bezug zu nehmen, wird das hier bezeichnete mobile und allgegenwärtige Rechnen genannt. Wie schon oben erwähnt, handelt es sich hierbei um eine Sammlung von Rechnern, welche die Fähigkeit haben, auf andere Rechner zuzugreifen, obwohl sie ihre physikalische Umgebung ständig wechseln. Die Kommunikation zwischen den Rechnern geschieht mithilfe einer Funktechnologie. So ist es nur notwendig, dass diese mobilen Rechner sich innerhalb der Funkreichweite befinden, um mit den restlichen Rechnern Verbindung aufbauen zu können. So stellen die einzelnen Rechner ihre Dienste zur

Verfügung, welche dann von den mobilen Rechnern genutzt werden können. Hierbei handelt es sich ebenfalls um ein verteiltes System, welches sich in der Vernetzung der einzelnen Rechner und der Eigenschaft der ständigen Mobilität der Clients unterscheidet.

### ISO-OSI Modell

Nun stellt sich die Frage, mit welchen heute verfügbaren Technologien das eben genannte allgegenwärtige verteilte System aufgebaut werden kann. Für die Beantwortung dieser Fragen wird das bewährte ISO-OSI Schichten Modell zu Hilfe genommen. Dabei werden nacheinander die einzelnen Schichten für sich selbst kurz erläutert und darauf folgend ein Bezug zwischen den Schichten und den zur Verfügung stehenden Technologien genommen. In der Abbildung 2.6 wird dieses Schichten Modell dargestellt.

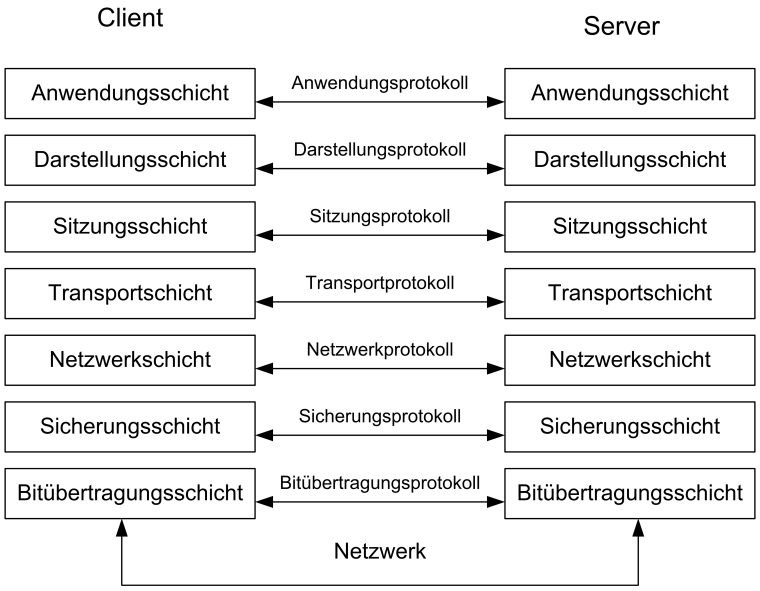


Abbildung 2.6: ISO-OSI Schichten Protokoll Modell (Tanenbaum und van Steen, 2002).

Dieses setzt sich aus insgesamt sieben unterschiedlichen Schichten zusammen, wobei jede Schicht eine entsprechende Schnittstelle zu der höher liegenden Schicht bereitstellt. Das ISO-OSI Schichtenmodell wurde 1992 von der *International Standards Organisation* (ISO) übernommen. Dieses Model soll bei der Entwicklung von Protokollstandards, die den Anforderungen eines offenen Systems genügen sollen, als Unterstützung dienen. Jede Schicht befasst sich jeweils nur mit einem Aspekt der Kommunikation.

**Bitübertragungsschicht** Die Aufgabe der Bitübertragungsschicht ist es, die Verbindungen und die jeweilige Hardware, in dem Netzwerk zu steuern. Dabei kann die Übermittlung der binären Signale auf verschiedene Weise geschehen. Für das oben genannte

Beispiel ist die Datenübertragung über Funk notwendig, sodass die Rechner (Clients) nicht mehr an einen Ort gebunden sein müssen.

**Sicherungsschicht** Die höher liegende Schicht ist die Sicherungsschicht. Diese ist für die Übertragungssteuerung von Dateneinheiten zwischen den Kommunikationsknoten zuständig. Dabei findet eine Übertragungsfehlererkennung und -korrektur statt.

**Netzwerkschicht** Nachfolgend kommt die Netzwerkschicht, welche für das Erstellen der Kommunikationsrouten und die Datenpaketübertragung zuständig ist.

**Transportschicht** Die Transportschicht ist die unterste Schicht, auf der eine Nachricht und keine Datenpakete verarbeitet werden. So findet in dieser Schicht die Adressierung der Nachrichten an die jeweilig für einen Prozess zugeordneten Kommunikationsports statt.

**Sitzungsschicht** Über der Transportschicht befindet sich die Sitzungsschicht, die für die Fehlererkennung auf der Nachrichtenebene zuständig ist. Sie stellt außerdem eine Erweiterung der Transportschicht dar.

**Darstellungsschicht** Die vorletzte Schicht ist die Darstellungsschicht. Diese hat als Aufgabe eine rechnerunabhängige Darstellung der Daten zu gewährleisten. Außerdem ist diese für die Verschlüsselung der Daten Zuständig.

**Anwendungsschicht** Die oberste Schicht ist die Anwendungsschicht. In dieser Schicht werden Protokolle zur Verfügung gestellt, die den Kommunikationsanforderungen bestimmter Anwendungen genügen.

### Allgegenwärtiges Verteiltes System

Es existieren viele Ausprägungen einer Middleware. Einige davon werden im nächsten Abschnitt analysiert. Hierdurch soll es möglich sein, am Ende eine Aussage treffen zu können, ob so eine Realisierung eines allgegenwärtigen verteilten Systems bereits heute möglich ist. Um bildhaft die Gemeinsamkeiten des allgegenwärtigen verteilten Systems mit dem hier vorgestellten allgemeinen Aufbau eines verteilten Systems zeigen zu können, wird in der Abbildung 2.7 eine Skizze des Aufbaus dargestellt.

Wie zu erkennen ist, weist der Aufbau dieses Systems große Ähnlichkeit mit dem allgemeinen verteilten System auf. So ist für die Kommunikation auf der Transport- und Sitzungssicht die bereits beschriebene Middleware zuständig. Durch diese soll die Kommunikation plattform- und sprachunabhängig ablaufen können. Auf diese setzt dann die in dieser Arbeit zu entwickelnde mobile Multimedia Middleware auf. Diese stellt auf einer höheren Abstraktionsebene die Kommunikation zur Verfügung. Deren Aufgabe ist es für die Mobilität und Allgegenwärtigkeit zu sorgen, sodass der Kommunikationsaufbau und -abbau transparent, für den Benutzer automatisch, geregelt werden kann.

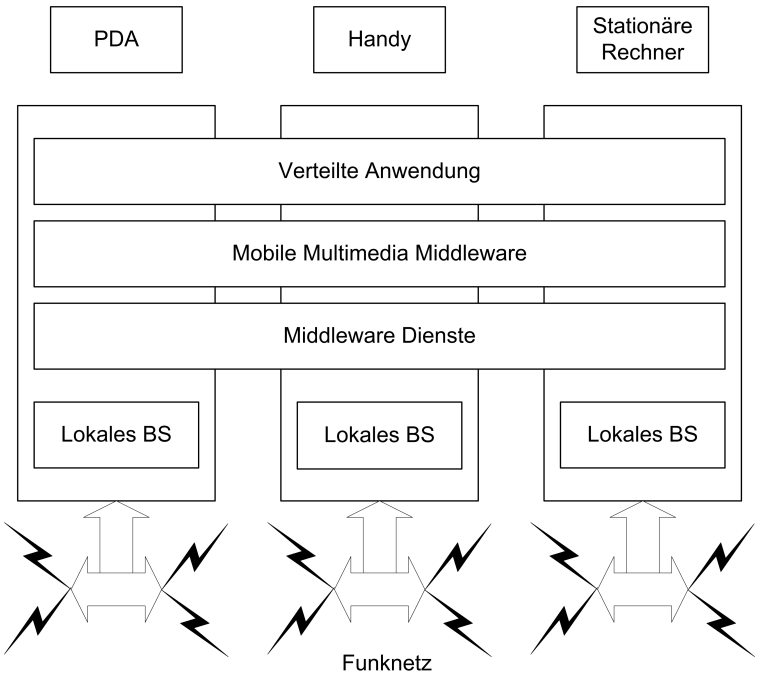


Abbildung 2.7: Allgemeine Architektur eines allgegenwärtigen verteilten Systems.

### 2.3.2 Hardware

#### Funktechnologie

Wie weiter oben bereits erwähnt, ist die Kommunikation bei einem allgegenwärtigen verteilten System mithilfe einer Funktechnologie zu erreichen. Heute gibt es verschiedene Funktechnologien, die die notwendige Funkverbindung bereitstellen können. So gibt es z.B. Bluetooth, Home RF, WiFi, IrDA und DECT. Diese unterscheiden sich im Preis, Datendurchsatz, Reichweite und der Anzahl der Kommunikationsteilnehmer von einander (s.z.B. Schlichting, 2003). Dabei stellen diese Standards jeweilige Protokolle zur Verfügung, die die Protokollschichten des ISO-OSI Modells entsprechen. Diese sind unterschiedlich mehr oder weniger in der Anwendungs-, Darstellungs- und Sitzungsschicht auf jeweilige Einsatzdomänen spezialisiert.

### 2.3.3 Middleware: Überblick

Um aber ein offenes, allgegenwärtiges verteiltes System ermöglichen zu können, wird in der Darstellungs- und Sitzungsschicht ein allgemeines Konzept gesucht, welches eine rechnerunabhängige Darstellung der Daten ermöglicht. Hier kommt die weiter oben angesprochene Middleware zum Einsatz. Diese stellt eine rechnerunabhängige Darstellung der Daten bereit und behandelt gleichzeitig die Probleme, die durch die Sitzungsschicht geregelt werden.

Wird diese durch die in ISO-OSI Schichtenmodel enthaltene Sitzungs- und Darstellungsschicht ersetzt, so bekommt man auf der höchsten Ebene ein unabhängiges Kommunikationsprotokoll. Hierdurch ist unabhängig von den im allgegenwärtigen verteilten System vorhandenen unterschiedlichen Rechnern, eine Kommunikation möglich (s.z.B. Heinrich, 2003).

**RPC - basiert**

In diesem Abschnitt sollen einige RPC basierte Middleware Implementierungen kurz beleuchtet werden. Dabei wird weniger versucht, auf jedes einzelne Detail einzugehen. Vielmehr soll hier ein Überblick über die verfügbare Middleware gegeben werden. Im Einzelnen werden die Middleware Abstraktionen RPC, XML-RPC sowie SOAP analysiert. Durch diese Betrachtung sollen dann Schlüsse gezogen werden, welche bei der Realisierung der Software helfen sollen.

**RPC** Einer der ersten Ansätze ist wohl der *Remote Procedure Call* (RPC). Dieses erlaubt einen Methodenaufruf zu tätigen, obwohl sich die Implementierung der aufgerufenen Methode auf einem anderen Rechner befindet. Dabei wird der aufzurufende Prozess suspendiert, bis das Ergebnis der Methode von dem anderen Rechner zurückgesendet wird. Es wird die Kommunikation mit dem entfernten Rechner, auf dem die Implementierung der Prozedur residiert, für den Anwendungsentwickler verborgen.

Zusätzlich bietet RPC asynchronen Methodenaufruf. Hierdurch wird die in dem Server vorhandene Methodenimplementierung aufgerufen, der aufzurufende Prozess wartet aber nur solange bis der Server den Methodenaufruf akzeptiert. Dieses Vorgehen wird dann benötigt, wenn eine Methode aufgerufen wird, diese aber kein Ergebnis zurückliefert. Viele der heute anzutreffenden verteilten Systeme und die dafür eingesetzte Middleware basieren auf RPC. So stellt Sun ihren RPC System bereit. Bei Microsoft heißt dieser *Distributed Computing*. Für weitere Information (s.z.B. Tanenbaum und van Steen, 2002).

**XML - RPC** XML-RPC ist nichts anderes als ein entfernter Methodenaufruf, welcher als Transportprotokoll HTTP für XML kodierte Daten benutzt. XML-RPC ist konstruiert worden, so einfach wie nur möglich zu sein. Zugleich aber erlaubt es komplexe Datenstrukturen zu transportieren, verarbeiten und zurückzugeben (UserLand Software, Inc., 2004). In der Abbildung 2.8 wird die Architektur von XML-RPC bildlich verdeutlicht.

**SOAP** SOAP ist ein Protokoll, welches dazu verwendet wird, entfernte Methodeaufrufe zu tätigen und Datenaustausch durchzuführen. Die Grundlage von SOAP stellen andere Standards, wie XML und verschiedene Internetprotokolle dar. XML wird für die Datenrepräsentation benutzt, die Internetprotokolle wie HTTP und TCP/IP für die Übertragung der Nachrichten (W3C, 2004a).

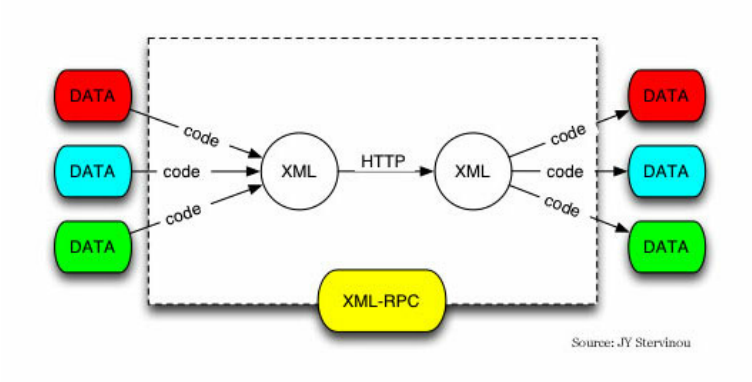


Abbildung 2.8: XML-RPC: Allgemeine Architektur (UserLand Software, Inc., 2004).

**Objekt - basiert**

Seit geraumer Zeit wird in der Anwendungsentwicklung die prozedurale Programmierweise durch die objektorientierte ersetzt. Der objektorientierte Ansatz verdankt seinen Erfolg unter anderem den *Information Hiding* (Parnas, 1972), durch welchen die internen Einzelheiten des Objekts nach außen verborgen bleiben. Hierdurch ist der Austausch eines Objekts relativ einfach, soweit sich die Schnittstelle nicht ändert.

So wurde ebenfalls Middleware entwickelt, die die objektorientierte Sichtweise unterstützt und somit die Vorteile des objektorientierten Ansatzes sich zu nutzen gemacht haben. Zusätzlich unterscheidet sich dieser Ansatz von dem RPC-System in der systemweiten eindeutigen Objektreferenz. So kann diese Objektreferenz dazu benutzt werden, um Objekte anderen Prozessen auf anderen Rechnern zu Verfügung zu stellen. Diese kann beispielsweise in dem Parameter der aufgerufenen Methode übergeben werden. Hierdurch wird die Transparenz der Verteilung im Vergleich zu RPC zusätzlich gesteigert. In diesem Umfeld kann *Java RMI*, *DCOM*, *CORBA* als Beispiel genannt werden.

**RMI** Java RMI (*Remote Methode Invocation*) unterstützt eine sehr einfache Kommunikation zwischen den Java-Objekten, die in den unterschiedlichen Java Laufzeitumgebungen existieren. Für den Anwendungsentwickler ist so gut wie kein Unterschied festzustellen, sodass dieser nicht erkennen kann, ob er gerade mit verteilten oder lokalen Objekten arbeitet. So ist durch Java RMI eine sehr intuitive Entwicklung der verteilten Anwendung gegeben. Zusätzlich ist hier ebenfalls die Verteilung in einem sehr großen Grad für den Anwendungsentwickler transparent gemacht worden. Dieser Ansatz hat aber gleichzeitig ein kleines Manko. Bei der Verwendung von Java RMI macht man sich von der Java-Programmiersprache abhängig. Diese Tatsache schränkt ein allgegenwärtiges verteiltes System in seiner Offenheit ein, sodass dieser Ansatz nur bedingt einsetzbar ist.

**CORBA** *CORBA* stellt ein Akronym für *Common Object Request Broker Architecture* dar. Diese Architektur ermöglicht eine Rechnerkommunikation, die auf verteilten Objekten basiert. Für die Kommunikation wird das IIOP-Protokoll benutzt, wodurch jede CORBA basierende Anwendung, unabhängig von der Implementierungssprache, Betriebssystem, Rechner und Netzwerk hinweg mit anderen CORBA Anwendung kommunizieren kann. Für weitere Informationen siehe (OMG, 2004a)

**DCOM** Von Microsoft wird ein objektorientiertes verteiltes System *DCOM* zu Verfügung gestellt, welches auf dem *Component Object Model* basiert und ebenfalls sprachunabhängig ist. Im Gegensatz zu COM, ermöglicht Distributed COM die Zusammenarbeit von Programmkomponenten nicht nur auf einem System sondern auf beliebig vielen.

DCOM hat aber den Nachteil, dass es speziell auf das von Microsoft entwickelte Betriebssystem Windows ausgelegt ist und somit mit diesem sehr stark gekoppelt ist. Diese Eigenschaft, also die Plattformabhängigkeit, schließt das Konzept für die weitere Betrachtung aus, da diese für das allgegenwärtige verteilte System von fundamentaler Bedeutung ist. Für weitere Informationen wird auf Tanenbaum und van Steen (2002) verwiesen.

#### Nachrichten - basiert

Abgesehen von dem RPC-Ansatz gibt es noch eine nachrichtenorientierte Kommunikation. Bei den oben beschriebenen Ansätzen wird vorausgesetzt, dass die Rechner, die die Implementierung der Methoden, bzw. der Objekte zur Verfügung stellen, bei dem Aufruf im Betrieb sind. Außerdem setzen die Server voraus, dass die Clients, die das Ergebnis der Berechnung angefordert haben, bei der Rückgabe ebenfalls betriebsbereit sind. Ist eine der Kommunikationsseiten nicht einsatzbereit, so ist eine Kommunikation nicht möglich. Dieses ist sozusagen ein Fehlerzustand.

Der nachrichtenbasierte Ansatz beschreibt ein System, welches eine Kommunikation ermöglicht, bei der zur Kommunikationszeit keine Notwendigkeit besteht, dass beide Kommunikationspartner betriebsbereit sein müssen. So kann der Client Anfragen an den Server stellen, obwohl dieser nicht betriebsbereit ist. Diese Anfragen werden dann in einem Puffer zwischengespeichert, bis der Server gestartet wird und diese abholt. Der Server kann dann die Anfrage bearbeiten und das Ergebnis zurücksenden, wobei die Nachricht ebenfalls in einem Puffer gespeichert wird, wenn der Client nicht laufen sollte. Die Kommunikation kann asynchron und synchron durchgeführt werden. Die meist genutzte nachrichtenorientierte Kommunikationsart ist die persistente asynchrone Kommunikation. Diese wird z.B. bei dem heute benutzen E-Mail Programmen benutzt.

Da der mobile Client die Eigenschaft besitzt, seinen Standort ständig zu wechseln und die Dienste, die er abrufen will, ortsrelevant sind, ist der Einsatz der nachrichtenorientierten Kommunikation in diesem Fall weniger sinnvoll. In der Natur eines lokalen Dienstes liegt es, die Problemstellung des Benutzers sofort an dem Ort, wo sich dieser befindet, zu be-

arbeiten. Eine verzögerte Bearbeitung der Benutzeranfragen macht nur in wenigen Fällen Sinn, sodass dieser Ansatz nicht weiter betrachtet wird. Für weitere Informationen wird auf Tanenbaum und van Steen (2002) verwiesen.

#### Service - basiert

**Web Services** Ein XML *Web Service* ist eine programmierbare Einheit, welche bestimmte Elemente einer Funktionalität wie Anwendungslogik liefert und diese für alle ungleichartigen Systeme, durch Benutzung der allgegenwärtigen Internetstandards, wie XML und HTTP, zugänglich macht. Web Services hängen sehr von der breiten Akzeptanz von XML und anderer Internetstandards ab. Diese ermöglichen die Erstellung einer Infrastruktur für die Unterstützung der Interoperabilität zwischen den Anwendungen auf einer Ebene, die viele Probleme, die vorher nicht lösbar waren, durch diese lösbar gemacht werden sollen.

Ein *Web Service* kann intern in einer Anwendung benutzt werden, kann aber auch nach außen über das Internet veröffentlicht werden, sodass dieser von beliebiger Anzahl von Anwendungen genutzt werden kann. Weil dieser durch eine standardisierte Schnittstelle zugänglich gemacht wurde, ermöglicht ein *Web Service* die Zusammenarbeit von heterogenen Systemen. Für weitere Informationen siehe (W3C, 2004b).

Wie eben schon erwähnt, werden durch *Web Services* neue Strukturierungsmöglichkeiten für ein verteiltes System geschaffen. So wird durch *Web Services* die Realisierung der *Service Oriented Achitecture* (SOA) ermöglicht. Diese beschreibt ein Strukturmodel für verteilte Anwendungen. Die primären Merkmale der SOA sind Dienste, welche benutzt werden um die großen Anwendungen in kleine separate Module zu teilen und diese durch entsprechende Zusammensetzungsmechanismen zu großen Anwendung integriert werden. Ein weiteres Kennzeichen der SOA ist, dass diese sich aus drei primären Kommunikationsteilnehmern zusammensetzt, den Dienstbereitsteller, den Dienstnutzer und der Dienstregistrierungsstelle. Für weitere Details siehe (IBM, 2004) und (SOA, 2004).

**JINI** JINI (*Java Intelligent Network Infrastructure*) ist eine Netzwerkarchitektur, für die es hochgradig wichtig ist, bei Konstruktion von verteilten Systemen das Verhältnis zwischen Veränderung und Komplexität der Wechselwirkungen innerhalb und zwischen den Netzwerken abzuschirmen. JINI Technologie liefert eine flexible Infrastruktur für Belieferungsdienste und die Durchführung spontanen Interaktionen zwischen den Clients und den Diensten ohne Berücksichtigung ihrer Hardware und Softwareimplementationen. So stellt JINI einen einfachen Satz von Java-Klassen und Diensten zur Verfügung, die Geräten und Diensten eine nahtlose Interaktion miteinander ermöglicht und macht diese ebenfalls zu einer für SOA geeigneten Plattform. Für weitere Informationen siehe (SUN, 2004b).

#### Kommunikationsprotokolle

**XML** Um den Einsatz von XML in dem betrachteten Anwendungsfall richtig einschätzen zu können und somit eine bestmögliche Alternative wählen zu können, wird eine kurze Beschreibung der Technologie folgen. Für die genauere Beschreibung der Syntax wird hier auf die reichlich vorhandene Literatur verwiesen (s.z.B. Doss, 2000) oder (vgl. Vonhoegen, 2004).

XML ist eine Weiterentwicklung von SGML. Dabei wurden viele der wenig benutzten Teile von SGML einfach weggelassen und einige internetspezifische Merkmale hinzugefügt. Die XML-Technologie wurde für das Web entworfen und beseitigt einen Mangel des Webs und speziell den von HTML. Gemeint ist die unzureichende Trennung zwischen dem Inhalt und der Präsentation. Hinzu kommt noch, dass eine inhaltliche Charakterisierung in den Dokumenten nicht möglich ist. XML-Daten stehen für sich und sind von ihrem Anwendungs- und Einsatzkontext losgelöst. Hier kurz einige Aspekte, die für XML-Daten gelten:

- Nachrichten, die mit Hilfe von XML formatiert sind, enthalten Strukturinformationen. Dies erleichtert deutlich das Analysieren und Weiterverarbeiten der XML-Nachrichten.
- Mit Hilfe von XML-Tags erhalten Dokumente eine gewisse Struktur, welche dann gezielte Zugriffe erlauben, die erheblich über eine strukturlose Volltextsuche nach einem bestimmten Textmuster hinausgehen.
- Da in XML eine Trennung zwischen Inhalt und Layout vorliegt, können XML Dokumente durch verschiedene Konvertierungs- und Layoutwerkzeuge in die jeweilige Darstellung z.B. HTML, SGML, WML usw. gebracht werden.
- Durch Dokumenttypdefinition bietet XML eine hierarchische Dokumentenstruktur, wodurch es möglich ist, Dokumente in unterschiedliche Teile zu zerlegen. Dies erleichtert dann enorm ein Dokumentmanagement. Hinzu kommt noch die Möglichkeit der Validierung des Dokuments, die durch die DTD geboten wird. Dadurch gewinnt man ein höheres Maß an Robustheit, wodurch die spätere Validierung der Daten in der Anwendung deutlich schrumpft.

Durch XML werden die Dokumente der realen Welt auf ein computerlesbares, hierarchisches, vor allem unabhängiges Datenschema reduziert. In der Form reduzierte und codierte Daten können gespeichert, bearbeitet, durchsucht, übertragen, angezeigt und ausgedruckt werden. Dokumente weisen eine hierarchische Struktur auf. Dadurch können sie in Komponenten aufgegliedert werden, welche ihrerseits auch wieder aus verschiedenen Komponenten bestehen. In XML werden diese logischen Komponenten, in die ein Dokument zerlegt werden kann, *Elemente* genannt. Elemente können andere Elemente enthalten oder den eigentlichen Text des Dokuments, der in XML als Zeichendaten bezeichnet wird. Durch Start- und Ende-Tag oder durch einen leeren Element-Tag sind die Elemente begrenzt. Da dieser hierarchische Aufbau eine baumartige Struktur aufweist, wird das Element, das alle anderen

Elemente enthält, als Wurzelelement oder Dokumentelement bezeichnet. Das Wurzelelement enthält Unterelemente, welche wiederum Unterelemente enthalten können. Enthält ein Unterelement kein weiteres Unterelement, wird dieser als Blatt bezeichnet. Dem Element können zusätzliche Eigenschaften oder andere charakteristische Informationen zugeordnet werden, die als Attribute bezeichnet werden. In Attributen sind ergänzende Daten über Elemente oder Elementeinhalt enthalten. Attribute haben Namen und Werte.

**XLT** Für die Darstellung von XML-Dokumenten wird Extensible Stylesheet Language kurz XSL verwendet. Mithilfe dieser Sprache wird in einem separaten Stylesheet nicht nur die Ausgabe, sondern allgemein die Beschreibung einer Transformation von XML in ein anderes Format definiert. Durch den Einsatz verschiedener Stylesheets lassen sich aus einer einzigen Datenquelle, einem XML-Dokument, sehr unterschiedliche Ansichten definieren. Bei der Konvertierung wird zwischen zwei Varianten unterschieden:

- Die Konvertierung des XML-Dokuments findet auf dem Server statt. Zu dem Client wird eine fertige HTML-Datei geschickt.
- Es wird das XML-Dokument und das Stylesheet an dem Client übertragen, der die Konvertierung selbst vornimmt.

### 2.3.4 Middleware: Genauere Betrachtung

#### CORBA

Ganz allgemein stellt CORBA eine Standardarchitektur für verteilte Objektsysteme dar. Dieser Standard ermöglicht Anwendungen zu schreiben, die aus einer Sammlung verteilter, heterogener und miteinander kooperierender Objekte besteht. Heterogen bedeutet in diesem Fall:

- Die verteilten Anwendungsobjekte müssen nicht alle in einer- und derselben Programmiersprache geschrieben sein.
- Sie können unter verschiedenen Betriebssystemen oder verschiedenen Rechnerarchitekturen laufen.
- Diese müssen nicht alle zur Entwicklungszeit von derselben Gruppe von Entwicklern erstellt worden sein.

Eine der wichtigsten Bestandteile der CORBA-Architektur ist der *Object Request Broker*, abgekürzt ORB. Deren Hauptaufgabe besteht darin, verteilte Objekte transparent miteinander kommunizieren lassen. Außerdem erlaubt ein ORB das Auffinden anderer, in dem verteilten System befindlichen, Objekte. Aus diesen beiden Gründen muss auf jedem Rechner, auf dem CORBA-Objekte residieren, ein ORB laufen.

Um Aufrufe von Objektmethoden über Rechnergrenzen hinweg zu ermöglichen, müssen die einzelnen ORBs dieser Rechner miteinander kommunizieren können. Diese Kommunikation wird mittels eines standardisierten *GIOP*[2] Protokolls bzw. speziell, auf das Internet abgestimmten *IIOP*[3] Protokolls, ermöglicht.

**Kommunikation zwischen den Objekten** Angenommen, es existiert irgendwo im Netz auf einem entfernten Server eine Objektimplementierung. Ein Client, in diesem Fall der PDA, möchte die Methoden nutzen, die dieses Objekt bereitstellt. Zu diesem Zweck übergibt er die entsprechenden Anfragen an den ORB, der die eigentliche Kommunikation erledigt. Die Abbildung 2.9 zeigt den oben geschilderten Ablauf.

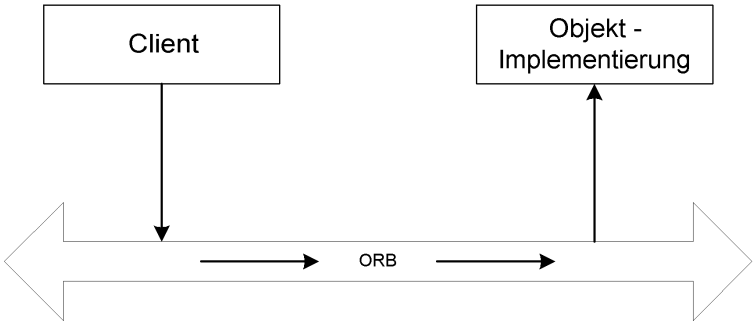


Abbildung 2.9: Grobe Struktur der Client-Objektkommunikation.

Wie aus Abbildung 2.9 hervorgeht, spielt der ORB in der Kommunikationskette eine wichtige Rolle. Aus diesem Grund wird hier dieser Teil der CORBA-Architektur zum besseren Verständnis des Kommunikationsablaufs näher beschrieben. Neben den generellen Komponenten, die im ORB für jede Anwendung verfügbar sind, werden weitere anwendungsspezifische Teile benötigt. Die einzelnen Teile kann man nach Client- und Serverseite unterteilen. Auf der Clientseite stehen Folgende zur Verfügung:

- Stub
- Dynamic Invocation Inteface
- ORB Interface

Auf der Serverseite stehen zur Verfügung:

- Skeleton
- Dynamic Skeleton Interface

[2] General Inter-ORB Protocol
[3] Internet Inter-ORB Protocol

- Object Adapter
- Implementation Repository
- Orb Interface

In Abbildung 2.10 werden die oben genannten Komponenten und dessen Zusammenspiel grafisch dargestellt.

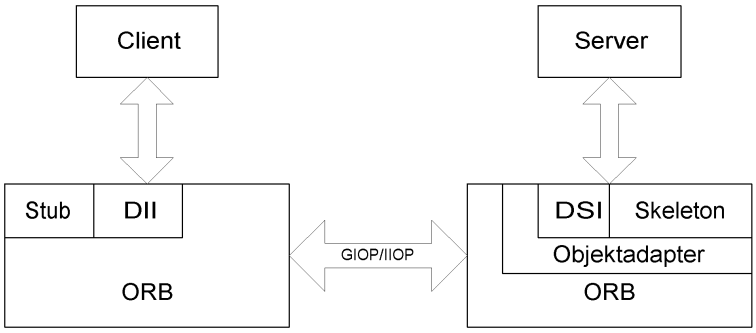


Abbildung 2.10: Zusammenspiel der einzelnen Komponenten.

**Stub und Skeleton** In der Netzwerkschicht ist die Einheit der Kommunikation die Nachricht. Als Nachricht wird eine Folge von Bytes bezeichnet, die von einem sendenden Rechner zu einem empfangenden übertragen wird. Die Funktionsweise der Versendung der Nachrichten auf der Netzwerkebene soll aber für den Client transparent sein, sodass der die Methoden auf ihm bekannte Weise aufrufen kann.

Diese eben geschilderte Anforderung wird mithilfe der Stubs und Skeletons ermöglicht. Ein Stub stellt die Verbindung zwischen Client und ORB dar. Seine Aufgabe besteht darin die Eingabeparameter zu verpacken und die Rückgabeparameter sowie den Rückgabewert zu entpacken. Auf der anderen Seite, gemeint ist die Serverseite, verbindet der Skeleton das Objekt mit dem ORB. Der Skeleton ist für den Aufruf der Schnittstelle des Serverobjekts, das Entpacken der Eingabeparameter, das Verpacken der Rückgabeparameter und des Rückgabewerts zuständig. Dieses Prinzip bezeichnet man als *statischen Methodenaufruf*, weil die Stubs und Skeletons zur Übersetzungszeit implementiert sein müssen, sodass zur Laufzeit keine Änderungen der Objektschnittstelle mehr möglich sind.

**Dynamic Interface und Repository** Das eben besprochene Prinzip ist beschränkt auf die in einer Applikation zur Übersetzungszeit bekannte Schnittstelle. CORBA ist jedoch ebenfalls auf die dynamische Definition und Verwendung entfernter Objekte ausgelegt. Um das dynamische Verhalten unterstützen zu können, sind in der CORBA-Architektur auf der Clientseite das *Dynamic Invocation Interface* (DII) samt dem *Interface Repository* (IFR) und auf

der Serverseite das *Dynamic Skeleton Interface* (DSI) und das *Implementation Repository* (IR) vorgesehen.

Mithilfe des DII können Methodenaufrufe zur Laufzeit dynamisch konstruiert werden. Damit der Client die Methoden benutzen kann, benötigt er Informationen über die Schnittstelle des aufzurufenden entfernten Objekts. Diese Information bekommt es über das IR, in dem die Namen und Parameter der Methoden des registrierten Objekts gespeichert sind.

Auf der Serverseite erlaubt es das DSI, Objekte zu unterstützen, die keine IDL - Beschreibung besitzen. Aus diesem Grund müssen sich die Objekte zur Laufzeit beim DSI registrieren, sodass die Information in der Datenbank des IR ablegt werden kann. Wenn Anfragen für Objekte beim DSI eintreffen, dann kann dieses aufgrund der Information im IR feststellen, ob das angesprochene Objekt auf diesem Server verfügbar ist.

**Eigenschaften** Hier werden alle Eigenschaften zusammenfassend genannt, um einen besseren Überblick über CORBA zu bekommen:

- Verteilungstransparenz: Der Zugriff auf lokale und ferne Ressourcen erfolgt in derselben Weise. Dadurch ist es bei der Anwendungsentwicklung nicht nötig sich mit dem komplizierten Problem der Kommunikation zwischen Rechnern und Prozessen auseinander setzen zu müssen.

- Ortstransparenz: Der Zugriff auf Ressourcen ist ohne Kenntnis des Ortes, an dem sich die Ressourcen befinden, durchführbar. Für das Auffinden der Objekte ist der ORB und der *Name Service*[4] zuständig. Der Client muss nur den Namen bzw. bei Nutzung des *Trade Service* nur Eigenschaften des gewünschten Dienstes kennen.

- Statische und dynamische Aufrufe: Prinzipiell müssen die Objektschnittstellen zur Zeit der Übersetzung der Anwendung nicht bekannt sein. Zur Laufzeit können neue Objekte mit neuen Schnittstellenbeschreibungen ins System kommen und ihre Dienste für Clients verfügbar machen.

- Unabhängigkeit von einer bestimmten Programmiersprache: CORBA ist an keine konkrete Programmiersprache gebunden. Dadurch ist es möglich, Lösungen für jede gängige Sprache zu entwickeln. Die auf der jeweiligen Basis entwickelten Anwendungsteile, können trotzdem miteinander kooperieren, sodass eine größtmögliche Flexibilität gegeben ist.

- Die OMG hatte in der Entwicklungsphase die Interoperabilität als eine der wichtigsten Zieleigenschaften betrachtet, sodass CORBA auf verschiedenen Plattformen eingesetzt werden kann.

[4] Einer der *Common Facilities*: Dienst zur Auffindung und Verwaltung der registrierten Objekte

- Verfügbarkeit von vielen vordefinierten Diensten: CORBA verfügt über einige zusätzliche Dienste, die den Anwendungsentwickler viele der Standardaufgaben abnehmen.

### Web Services

**Überblick** Was ist ein Web Service? Um diese Frage zu beantworten, wird die in der (W3C, 2004c) gegebene Definition zu Hilfe genommen:

> *A Web service is a software system, whose public interface and bindings are defined and described using XML. Its definition can be discovered by other software systems. These systems may then interact with the Web services in a manner prescribed by its definition, using XML based messages by Internet protocols.*

Somit ist hier eine Softwarekomponente gemeint, die ihre Schnittstelle, durch die man auf diese zugreifen kann, in einer darstellungsneutralen Form mit Hilfe von XML beschreibt und nach außen offen legt, sodass diese für alle sichtbar ist. Hierdurch sollen andere Softwarekomponenten die Möglichkeit haben diese Schnittstellendefinition zu erkennen, um dadurch mit dieser zu kommunizieren und deren zur Verfügung gestellten Methoden auszuführen. Der Austausch der Daten zwischen den Softwarekomponenten, soll ebenfalls in einer neutralen Darstellungsform, dass das XML liefert, von statten gehen. So ist eine allgemeine Forderung in der Definition die Offenheit.

Aus der oben zitierten Definition eines Web Service ergeben sich für diesen einige Eigenschaften. Diese sind:

- XML - Basierbarkeit
- Lose Bindung
- Abbildung auf Geschäftsmethoden
- Synchrone oder asynchrone Methodenausführung

So ist mit der XML-Basierbarkeit die Forderung nach der Verwendung von XML gemeint. Durch diese werden die durch Protokolle der Plattform und des Netzwerks entstehenden Bindungen eliminiert. Dadurch werden die Grenzen der verschiedenen Plattformen überwunden. Außerdem wird die Eigenschaft der allgemeinen Darstellung von Daten von XML mit übernommen. Dadurch können einfache sowie komplexe Daten zwischen den Kommunizierenden ausgetauscht werden.

Mit der losen Bindung ist hier die Entkopplung der Clientseite von dem von ihm benutzen Web Service gemeint. So ist der Client bei einer Änderung der Schnittstelle des Web Services nicht beeinträchtigt, sodass dieser ohne Hindernisse und irgendwelcher Anpassungen

weiterhin auf den Web Service zugreifen kann. Durch diese Eigenschaft wird die Integration der verschiedenen Softwarekomponenten deutlich vereinfacht.

Bei der Eigenschaft „Abbildung von Geschäftsmethoden“ handelt es sich um eine Sichtweise auf die von einen Web Service zur Verfügung gestellten Methoden. Bei einem Web Service sollen Geschäftsprozessmethoden zur Verfügung gestellt werden, welche eine Menge von Methoden in sich kapseln und somit ganze Geschäftsprozessdienste darstellen.

Ein Client, der auf einen Web Service zugreift, hat die Möglichkeit, an einen Web Service eine Anfrage synchron oder asynchron auszuführen. Bei der synchronen Ausführung wird der Client solange blockiert, bis der entsprechende Dienst die Anfrage beantwortet. Bei der asynchronen Ausführung kann der Client weitere Anfragen stellen und die erforderlichen Ergebnisse der Anfragen später bei dem Web Service abholen.

**Architektur** Die Web Services Technologie steht für eine Menge von mehreren anderen Technologien, durch welche die eigentlichen Web Services ermöglicht werden. Nicht alle Technologien sind von der Spezifikation vorgeschrieben. Es haben sich aber inzwischen einige in dem Umfeld etabliert. Zu erwarten ist aber, dass bis zur Vollendung der Spezifikation einige Technologien wechseln werden.

Zurzeit werden hauptsächlich drei Technologien benutzt. Diese sind ebenfalls der Bestandteil der Web Services Spezifikation. Das sind:

- *Simple Object Access Protocol* (SOAP)
- *Web Service Description Language* (WSDL)
- *Universal Description, Discovery and Intergration* (UDDI)

Bei SOAP handelt es sich um einen Standard, der eine Verpackungsstruktur für XML-Dokumente bereitstellt, welche dann über Internettechnologien wie HTTP transportiert werden können.

Bei der WSDL handelt es sich ebenfalls um eine XML-Technologie, welche für die standardisierte Beschreibung der Schnittstellen der Web Services verwendet wird. Mit Hilfe von WSDL werden die Ein- und Ausgabeparameter, Struktur der Methoden, sowie das Dienstprotokoll beschrieben, wodurch die Clients die Schnittstelle des Web Services verstehen und mit diesen kommunizieren können.

Bei der UDDI-Technologie handelt es sich um eine weltweite Registrierungsstelle von Web Services. Diese dient dazu, neue Web Services für die Außenwelt sichtbar zu machen, bereits Bestehende mit entsprechenden Suchkriterien wie Name, Kategorie zu finden und die Spezifikationen und Web Services Zugangspunkte darzustellen.

Die Abhängigkeiten zwischen den eben erwähnten Technologien SOAP, WSDL und UDDI sehen wie folgt aus:

1. Ein Client der auf einen speziellen Web Service zugreifen will, muss diesen zuerst finden. So richtet sich dieser als Erstes bei der Registrierungsstelle, bei der er eine Suchanfrage nach einen Web Service abgibt, der den entsprechenden Suchanforderungen wie Name, Kategorie, Spezifikation entspricht.

2. Ist ein Web Service gefunden worden, auf dem die entsprechenden Suchanforderungen zutreffen, so holt sich der Client aus der Registrierungsstelle (UDDI) die Informationen über die Lage des Web Services. Diese Information steht in Form eines WSDL-Dokuments zur Verfügung. Dieses Dokument enthält Informationen über den Web Service, die dazu notwendig sind, um mit diesem eine Verbindung aufbauen zu können. Außerdem wird in diesem beschrieben, wie die Anfragen formatiert werden müssen, damit der Web Service sie versteht.

3. Nachdem der Client das WSDL-Dokument und somit die Schnittstelle des Web Services richtig erkannt hat, erzeugt dieser eine XML-Schema konforme SOAP-Nachricht, welche er dann an den Host schickt, auf dem sich der entsprechende Web Service befindet.

Diese eben geschilderte Abhängigkeit wird noch mal in der Abbildung 2.11 dargestellt.

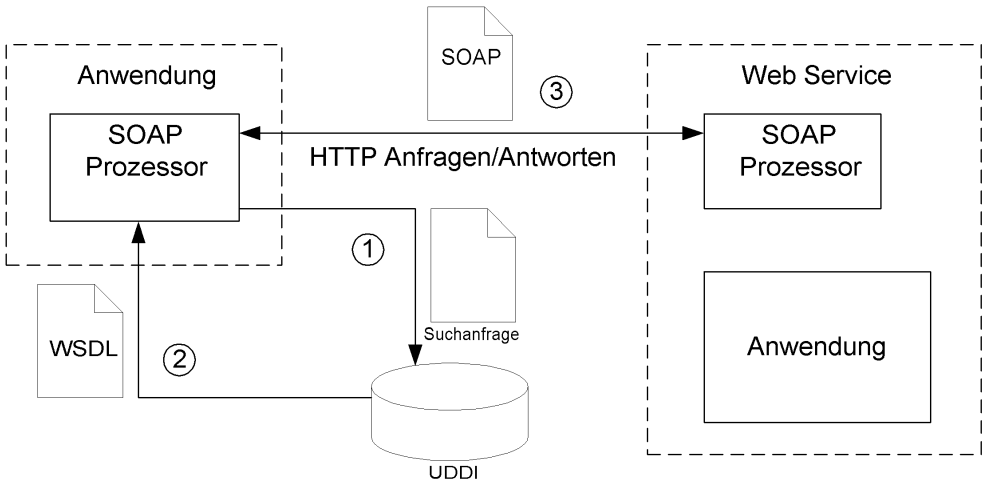


Abbildung 2.11: Abhängigkeit zwischen den einzelnen Web Services Technologien.

Das Web Service Konzept, welches die Dienstorientierte Architektur (SOA) abbildet, kann dazu verwendet werden, um einfache Methodenaufrufe zu tätigen. Wie bereits aus der Bezeichnung der Web Services zu entnehmen ist, ist dieses Model eher dazu bestimmt, einen Dienst zur Verfügung zustellen, welcher eine ganze Ansammlung von Geschäftsvorgängen anbietet, die eine komplexe Geschäftsfunktionalität nachbilden. Unerheblich was im Endeffekt der Web Service am Ende bereitstellt, stellt dieser eine abgeschlossene Einheit dar, welche sich selbst beschreibt und seine Dienste in der Web Service Welt bereitstellt.

Web Services können aber auch selbst auf andere Web Services zugreifen. Dadurch können nicht nur einfache, sondern auch komplexe Web Services entstehen, die die Einzelnen in einer entsprechenden Geschäftslogik verbinden und somit auf einer anderen Geschäftsebene bereitstellen können. Bei der Kapselung ist theoretisch keine Grenze gesetzt, wodurch Dienste in verschiedensten Variationen angeboten werden können.

Durch die Offenheit, die das Web Service Modell offeriert, ermöglicht dieses den alten bereits bewährten Softwarekomponenten, diese Web Service fähig zu machen und in das Modell zu integrieren. Diese Eigenschaft dieser Technologie macht sie sehr attraktiv, da hierdurch im großen Maße die Wiederverwendung unterstützt wird.

**Anwendungsmöglichkeiten** Die Web Services versprechen plattformunabhängig zu sein, wodurch ein hohes Maß an Interoperabilität gegeben wird. Diese Interoperabilität ermöglicht eine neue Sichtweise auf Datenverarbeitung. So können alle erdenkliche Dienste entstehen, die Geschäftsmethoden für alle möglichen Geschäftsszenarien zur Verfügung stellen, auf die man dann über das Internet jederzeit zugreifen kann. So spricht man davon, dass das Web eine Wandlung zur Dienstorientiertheit vornehmen wird. Durch die Interoperabilität der Web Services Technologie sowie durch die Möglichkeit der Kapselung der einzelnen Dienste zu einem wird z.B. ein Problem, welches heute in dem Business-To-Customer Konzept auftaucht, beseitigt.

So entstehen immer mehr Internet Portale, die sich darauf spezialisieren, eine große Menge Anbieter einer speziellen Gruppe zusammen in Ihrem Portal für potenzielle Kunden bereitzustellen. Als Beispiel kann hier die Autoindustrie genannt werden. Diese produziert schon lange keine Einzelteile mehr, sondern kauft sie bei kleinen Fertigungsunternehmen ein. Bei dem heutigen Konkurrenzkampf zählt am Ende derjenige, der seine Produkte so günstig wie möglich und in bester Qualität liefert. Um also ihre Konkurrenzfähigkeit zu steigern, suchen die Automobilkonzerne Möglichkeiten eines Globalen Preis- und Qualitätsvergleiches. Diese wird durch die entsprechenden Internet Portale geboten. Diese haben aber mit dem Problem zu kämpfen, dass bei der großen Menge der Unternehmen, ebenfalls eine große Menge an Datendarstellungen vorhanden sind. Dadurch wird eine einheitliche Darstellung erheblich erschwert, wodurch ein Vergleich nur eingeschränkt automatisiert werden kann. Durch die Web Services und SOAP-Technologie wird die Kommunikation und Datendarstellung in einer neutralen Form vonstatten gehen, wodurch dieses Problem behoben wird.

Um an das in dieser Arbeit verfolgte Anwendungsszenario anzuschließen, wird ein Beispiel gegeben, welches die Vorteile schildert, die durch die Verwendung der Web Services Technologie entstehen. Ein Benutzer, der zum Beispiel einen mobilen PDA besitzt, kommt in die Reichweite eines Zugangspunktes[5], wodurch dieser sich mit dem ortsabhängigen Netzwerk verbinden kann. Beispielhaft kann ein Zug mit einem installierten Zugangspunkt so ein Bereich darstellen (s.o.). So steigt dieser in diesen Zug ein, welches offen gelegte Web Services für verschiedene mobile Geräte anbietet. Nach der Verbindung mit einer von dem Zug

[5] auch als Hotspot bekannt

zur Verfügung gestellten Funkschnittstelle, wird der PDA diesen nach geeigneten Diensten absuchen. Der PDA fragt bei der Registrierungsstelle nach einem geeigneten Dienst, lädt die geeignete Schnittstellenbeschreibung und greift auf diesen zu. So wurden in diesem kurzen Szenario gleich mehrere Problemstellungen überwunden, die Dienstauffindung, Interoperabilität bei der Kommunikation und die Darstellung des Dienstes auf den PDA. Die Dienstauffindung findet mithilfe der Registrierungsstelle statt, die Dienstkommunikation wird durch das beigelieferte WSDL-Dokument ermöglicht, die neutrale Darstellung der Information wird mithilfe der SOAP Technologie gegeben. Die richtige Darstellung wird durch das Auffinden des geeigneten Dienstes in der UDDI gewährleistet. So existiert für jede Gruppe von mobilen Geräten eine entsprechende Dienstimplementierung.

**Was ist SOA** Immer öfter wird von einer dienstorientierten Welt gesprochen, in der überall über ein Netzwerk Dienste verfügbar sind, die jede erdenkliche Leistung zur Verfügung stellen können. Die Dienstorientierte Architektur (SOA) ist ein Konzept, welches dieses Szenario ermöglichen soll. Diese wurde in (IBM, 2004) definiert. In dem Dokument werden Möglichkeiten dargestellt, wie man auf der Basis der Web Services Technologien, SOAP, WSDL und UDDI, eine dienstorientierte Welt ermöglichen kann. In (IBM, 2004) besagt die Definition, dass ein Web Service aus zwei Teilen besteht. Der Hauptbestandteil ist ein Dienst, mit dem zweiten Teil ist die Dienstbeschreibung gemeint. Die Dienstimplementierung kann in verschiedenen Sprachen geschehen. Einzige Anforderung, die an einen Dienst gestellt wird, ist die, dass dieser eine Möglichkeit anbieten muss, über ein Netzwerk erreichbar zu sein. Die Dienstbeschreibung stellt eine Möglichkeit dar, einen Dienst mit Hilfe von XML nach außen zu beschreiben. Diese Beschreibung muss inhaltlich standardisiert sein. Durch die Dienstbeschreibung wird eine Schnittstellenbeschreibung angeboten, wodurch dieser erreichbar wird.

**SOA Interaktion und Rollen** Die SOA Spezifikation stellt ebenfalls eine Beschreibung der Geschäftsprozesse dar, wie sie in dieser Architektur vorzufinden sind. Dabei werden drei Rollen definiert, die bei dem Szenario auftreten können. So finden Interaktionen zwischen:

- Dienstbereitsteller,
- Registrierungsstelle,
- Anfragenden.

Dabei werden folgende Aktivitäten identifiziert:

- Dienstregistrierung und Veröffentlichung,
- Dienstlokalisierung,

- Dienstbenutzung.

Unter einen Dienstbereitsteller ist der Besitzer und Bereitsteller des Web Services zu verstehen. In seiner Verantwortung stehen die Veröffentlichung des Web Services und die Bereitstellung der verschiedenen Zugriffsmethoden.

Die Dienstregistrierungsstelle hat die Aufgabe, Informationen über alle in ihr registrierten Web Services und deren Dienstbereitsteller zu verwalten. Es werden verschiedene Informationen zusammengetragen, um eine Möglichkeit einer Klassifizierung der Dienste vornehmen zu können. Zusätzlich werden von der Registrierungsstelle Suchfunktionen angeboten, die eine Suche auf gesammelten Metadaten erlauben.

Ein Anfragender wird in der SOA als ein Interessent erkannt, welcher Interesse an den verschiedenen Softwarekomponenten, die als Web Services verfügbar sind, hat und auf diese dann zugreift.

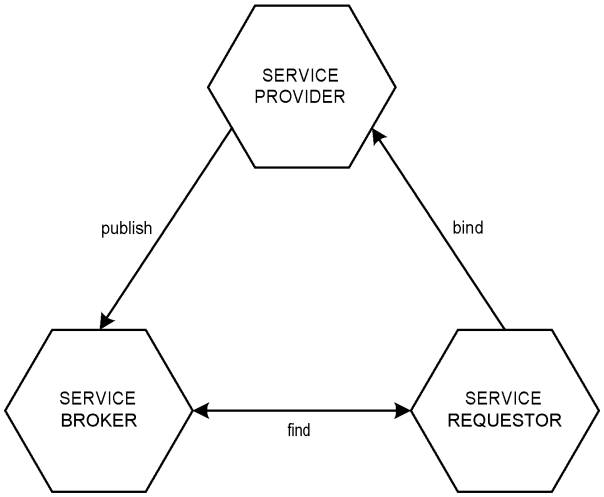


Abbildung 2.12: Interaktionen und Rollen in der SOA (IBM, 2004).

Die Interaktionen, die unter den drei Teilnehmern stattfinden, werden ebenfalls spezifiziert. Bei der Dienstveröffentlichung veröffentlichen die Dienstbereitsteller eigene Dienste, indem sie die entsprechenden Dienstinformationen in der Registrierungsstelle hinterlegen. Die Dienstveröffentlichung kann dabei direkt oder dynamisch geschehen.

Die Dienstlokalisierung beinhaltet einen Zugriff auf die Registrierungsstelle. Bei dieser werden Suchanfragen nach den gesuchten Diensten gestellt. Dabei werden verschiedene Eigenschaften der Dienste betrachtet, sowie deren eigene Schnittstellenbeschreibung, um deren unterstützte Kommunikationsprotokolle erkennen zu können.

Wird ein Dienst erfolgreich bei einer Registrierungsstelle gefunden und der Anfragende äußert den Wunsch mit diesen zu kommunizieren, wird diese Interaktion als Dienstnutzung erkannt. Alles was der Anfragende über die Schnittstelle des Dienstes wissen muss, findet er über die in der Registrierungsstelle enthaltenen, in XML geschriebenen Web Services

Beschreibungsdokumente. Die eben vorgestellten Teilnehmer und ihre Interaktionen werden in Abbildung 2.12 dargestellt.

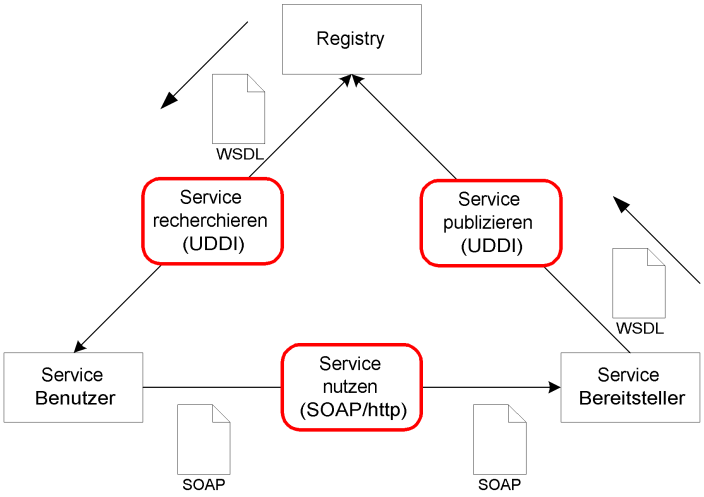


Abbildung 2.13: Anwendungsmodell von Web Services und deren Spezifikationen (Mörike, 2003).

Um einen Übergang von SOA auf Web Services vornehmen zu können, wird in der Abbildung 2.13 das Anwendungsmodell von Web Services, sowie der Einsatz der zugrunde liegenden Spezifikationen bildlich dargestellt.

**Was wird durch SOA erreicht** Bei der SOA werden in (Chappell und Jewell, 2003) zwei Perspektiven, die die Teilnehmer haben können, identifiziert. Diese unterscheiden sich durch die unterschiedlichsten Ziele, die diese in der SOA verfolgen. So werden hier einmal die Geschäfts- und Entwicklerperspektive genannt.

Aus der Geschäftsprozesssicht, stellt die SOA eine Architektur dar, welche einem teilnehmenden Dienstbereitsteller erlaubt, verschiedene Strategien bei der Distribution der verschiedenen Dienste zu verfolgen. So kann ein Unternehmen, welches in diesem Fall als ein Dienstbereitsteller identifiziert wird, unabhängige Dienste implementieren, pflegen und nach außen bereitstellen. SOA stellt aber auch Möglichkeiten für Unternehmen bereit, die sich nicht darauf spezialisiert haben, Dienste zu entwickeln und nach außen zu veröffentlichen. So kann ein Unternehmen, welches eine Geschäftslogikimplementierung für eigene Zwecke benutzt, welche für Andere von Interesse ist, diese als ein Dienst nach außen bereitstellen und an andere für entsprechendes Entgelt vermieten.

Es ist nicht notwendig Dienste zu implementieren, um bei der dienstorientierten Architektur als Dienstbereitsteller teilnehmen zu können. In der SOA kann ein Unternehmen ebenfalls als Dienstbereitsteller für bereits vorhandene Dienste, die zu einer noch nicht bestehenden Geschäftslogik zusammengefasst sind, fungieren. So greift dieses Unternehmen auf beste-

hende Dienste zu, fügt diese zusammen, sodass diese eine neue Geschäftslogik ergeben, was nach außen als ein neuer Dienst bereitgestellt wird.

Aus der Sicht des Dienstproviders stellt SOA also eine Rahmenumgebung für das Veröffentlichen verschiedener Softwarekomponenten dar. Durch die neue Vorgehensweise, die durch SOA vorgegeben wird, ergeben sich aber auch neue Probleme, die ein Dienstanbieter im Hinterkopf behalten muss. So ist ein nicht durchgehend verfügbarer Dienst für Andere nicht von großem Interesse. Also muss sich der Dienstbereitsteller darum kümmern, dass sein Dienst durch Lastenverteilung auf skalierbare Weise nach außen hoch verfügbar gemacht wird. Weiterhin sind Dienste dann interessant, wenn sie hoch effizient arbeiten. Außerdem wird in der SOA vorausgesetzt, dass ein Dienst als Teil von Geschäftsabläufen verwendet werden kann. Diese müssen also entsprechend darauf konzipiert sein. Eine nicht zu vernachlässigende Anforderung ist die Möglichkeit einer Versionskontrolle und Fernwartung.

Übernimmt ein Unternehmen die Rolle der Registrierungsinstanz, so ergeben sich in der SOA für diesen folgende Ziele: Ein Unternehmen verwaltet Dienste und deren Einträge und publiziert diese nach außen, wodurch Registrierungsgebühren für den Dienstbereitsteller anfallen. Eine Registrierungsinstanz kann aber in der SOA die Aufgabe übernehmen, andere Registrierungsstellen miteinander zu verbinden, sodass die Bekanntmachung eines Dienstes sich über mehrere Registrierungsstellen ausdehnt. Außerdem können in der SOA andere Arten von Registrierungsstellen vorgesehen werden. Somit können sich Unternehmen auf das Auffinden und Installieren von Dienstdokumentationen spezialisieren.

Für ein Unternehmen, welches an einen Dienst interessiert ist, also eine Rolle des Anfragenden übernimmt, stellt SOA eine Rahmenumgebung dar, die ihm erlaubt, gesuchte Dienste zu finden und auf diese zuzugreifen. So werden ihm Such- und Zugriffsmethoden zur Verfügung gestellt, welche den Anfragenden erlauben, die Problemstellung oder Teilprobleme, die er bearbeiten will, an veröffentlichte Dienste weiter zu geben und von diesen, gegen entsprechende Gebühr bearbeiten zu lassen.

Für einen Anwendungsentwickler stellt SOA ebenfalls neue Sichtweisen dar. So bietet diese Rahmenumgebung für verschiedene, für den Entwickler interessante Aspekte, die nach den in der SOA identifizierten Rollen unterteilt werden können:

- Dienstbereitsteller,
- Anfragender.

Aus der Sicht eines Entwicklers, der einen Dienst bereitstellen will, erlaubt die SOA eine unabhängige Implementierung eines Dienstes, deren Inbetriebnahme und Veröffentlichung. Dieser entwirft und beschreibt die Schnittstelle des Dienstes, stellt die Dienstimplementierung und nimmt den Dienst in Betrieb. Nach der Inbetriebnahme ist der Entwickler weiter für die Pflege und Verbesserung der Dienste zuständig. Die Veröffentlichung kann, muss aber nicht, ebenfalls von dem Entwickler übernommen werden. Diese Vorgänge sind von der anfragenden Seite entkoppelt.

Aus der Sicht des Entwicklers, der die anfragende Seite implementiert, stellt die SOA eine einfache Möglichkeit der Bindung an einen Dienst dar. Diese wird den Entwickler durch eine Vorlage ermöglicht, die die Dienstnutzung beschreibt. Sie enthält die Definition der Schnittstellen, sowie Anforderungen die an den Anfragenden gestellt werden, um den Dienst nutzen zu können.

### J2EE und .NET

In dem Zusammenhang mit Web Services Technologie, sollte etwas auf die heute aktuellen Entwicklungsplattformen eingegangen werden. Gemeint sind in diesem Fall die *J2EE* von *SUN* und die *.NET* Plattform von *Microsoft*. Diese beiden Architekturen versprechen ebenfalls, einen großen Teil in Richtung der Interoperabilität gemacht zu haben. Dies wird aber mehr oder weniger plattformintern gewährleistet. Es stellt sich an dieser Stelle die Frage, ob die Web Services Technologie mit diesen Technologien kombiniert werden können?

**Positionierung der beiden Plattformen** In der Web Service Spezifikation ist die Rede von einer Web Service Plattform. Diese ist eine Umgebung, welche einen oder mehrere Web Services enthält. Außerdem gehört ein SOAP-Server, UDDI als Registrierungsstelle, sowie Transaktions- und Sicherungsdienst zur dieser Plattform dazu. Außer den eben spezifizierten Plattformdiensten, gehören bei unterschiedlichen Anforderungen weitere wie Clustering und Stapelverarbeitung hinzu. Diese werden von den Web Services nicht zur Verfügung gestellt, legen aber die benötigten Schnittstellen offen, sodass die erforderlichen Dienste von außen gegeben werden können. Diese kann durch die Web Service Anbieter erbracht werden. J2EE und .NET werden als solche Anbieter erkannt. Sie besitzen die benötigten Konzepte um die erforderlichen Dienste, sowie die an diese gestellten Anforderungen für die Web Services zur Verfügung zu stellen, sodass die Kernfunktionalität der Web Service Plattform auf diesen aufbauen kann. Die Web Services Technologie bekommt in diesem Kontext die Aufgabe, alle auf der Plattform verfügbaren Dienste in einer standardisierten Form nach außen sichtbar zu machen und offen zu legen.

**J2EE: einfachere Dienste** So stellen die beiden Entwicklungsplattformen verschiedene Möglichkeiten bei der Unterstützung der Web Service Technologie. J2EE bietet der Web Service Technologie mehrere Ansatzpunkte für eine mögliche Erweiterung der Web Services Funktionalität. Durch die in der J2EE zur Verfügung gestellten *Servlet* Technologie, sowie darauf aufbauende JSP Technologie, wird eine Umgebung für Web Services geschaffen, die einfacher Natur ist. Servlets bzw. JSP (*Java Server Pages*) werden hier dazu verwendet, um auf der Senderseite Daten in die SOAP-Nachricht zu verpacken, auf der Empfängerseite diese wieder ordnungsgemäß aus der SOAP-Nachricht zu holen. Diese stellen einen Einstiegspunkt für die Web Service Nachrichten dar und leiten diese zu den entsprechenden J2EE Diensten weiter.

**J2EE: anspruchsvollere Dienste** Es werden aber nicht nur Web Services einer einfachen Natur unterstützt. So werden heute Dienste benötigt, die skalierbar, zuverlässig und hochverfügbar verteilte Geschäftslogik zur Verfügung stellen. Diese werden mithilfe der EJB Technologie ermöglicht. So wird bei solcher Architektur eine SOAP-Nachricht mittels eines Internetprotokolls von einem Client an einen Web-Server, auf dem sich ein SOAP-Prozessor befindet, gesendet. Diese wird ebenfalls durch ein Servlet implementiert. Nachdem der SOAP-Prozessor die SOAP-Nachricht geparst und die entsprechende EJB-Referenz geholt hat, schickt dieser die Anfrage weiter an die richtige EJB (*Enterprise Java Beans*). Diese agiert mit den angeforderten Ressourcen und schickt nach der Bearbeitung der Anfrage eine Antwort an den SOAP-Prozessor zurück. Der SOAP-Parser baut daraus eine gültige SOAP-Nachricht zusammen und sendet diese an den Client.

**J2EE: asynchrone Dienste** Ist eine lose gekoppelte, asynchron kommunizierte Anwendung, die als offener Web Service zur Verfügung gestellt werden soll, erforderlich, so stellt J2EE hierfür ebenfalls eine Möglichkeit zur Verfügung. *JMS* (*Java Message Service*) ist eine weitere Kommunikationstechnologie, die die J2EE Plattform anbietet. Die Vorgehensweise ist vergleichbar mit der der EJB's, unterscheidet sich nur in der Kommunikationsweise. Eine SOAP-Nachricht wird von dem Client wieder an den SOAP-Prozessor gesendet. Dieser parst sie, baut eine gültige JMS-Nachricht und leitet sie entsprechend der Vorgaben weiter an sein JMS Ziel. Ein entsprechendes Ziel, welches als Beispiel eine JMS-Queue sein kann, verarbeitet die Nachricht, indem sie die angeforderte Aufgabe bearbeitet. Nach der Bearbeitung sendet dieser, abhängig von der Übertragungsart, das Ergebnis an den Client zurück. Bei der Anfrage/Antwort-Übertragungsart, wird das Ergebnis zu dem SOAP-Prozessor zurückgesendet, welches diese in eine SOAP-Nachricht verpackt und an den Client, der die Anfrage gesendet hat, zurücksendet. Bei der einseitigen Übertragung wird keine Antwort von dem Client erwartet.

**.NET: einfache Dienste** Die .NET Plattform stellt ebenfalls einige Möglichkeiten für die Zusammenarbeit mit den Web Services zur Verfügung. So werden für den Aufbau und für das Parsen einer SOAP-Nachricht, die über ein Internetprotokoll versendet wird, *Active Server Pages* (ASP.NET) zur Verfügung gestellt. Active Server Pages sind mit der JSP-Technologie vergleichbar. Durch diese kann die SOAP-Nachricht entsprechend verarbeitet und weiter an die entsprechenden Dienste geleitet werden. In dieser Umgebung werden einfache Web Services verwaltet. Der Ablauf der beidseitigen Kommunikation ist analog zu der in der *J2EE*.

**.NET: anspruchsvollere Dienste** Erfordert ein Web Service eine ausgebaute, komponentenbasierte Architektur, so stellt die .NET Plattform .NET *Management Components* zu Verfügung. Diese ist eine EJB vergleichbare Technologie und wird für die Implementierung der Geschäftslogik verwendet. Für die Kommunikation wird weiterhin die ASP.NET Technologie benutzt. Ansonsten lässt sich zusammengefasst sagen, dass der Ablauf von der Erstellung

einer Anfrage, ihre Versendung, Verarbeitung und Beantwortung im Groben sehr mit dem oben bei J2EE geschilderten Ablauf vergleichbar ist. Dieser unterscheidet sich natürlich in den jeweiligen Einzelheiten, verfolgt aber ein ähnliches Prinzip.

**.NET: Sprachunabhängigkeit** Die .NET Plattform bietet ferner, zumindest in der Grundüberlegung, außer der Plattformunabhängigkeit ebenfalls eine Möglichkeit, die Web Services mit verschiedenen Sprachen zu implementieren, somit ist also eine Sprachunabhängigkeit vorhanden. Dabei sind im Genauen nicht die jeweiligen, seit langer Zeit auf dem Markt verfügbaren, Programmiersprachen gemeint. Vielmehr werden hier Programmiersprachen angeboten, die auf die .NET Plattform ausgerichtet wurden. Die Sprachunabhängigkeit, so wie sie z.B. bei CORBA vorzufinden ist, ist hier nur eingeschränkt vorhanden. Die bestehenden Programme, besonders solche, die direkt im Speicher Änderungen vornehmen können, Pointerfunktionalität verwenden und keine Garbage Collection benutzen, sondern manuell den Speicher verwalten, werden nur bedingt von der .NET Plattform unterstützt. Somit wird in dieser Betrachtung die von der .NET Plattform zugesicherte Sprachunabhängigkeit vernachlässigt. Es muss aber noch erwähnt werden, dass, wenn die Programmiersprachen und dafür entsprechende Compiler verbreiteter sein werden, die die jeweiligen Sprachdialekte vollständig auf die .NET Plattform portieren, diese Situation neu betrachtet werden müsste. Heute lässt sich über dieses Feature nichts Konkretes sagen, da die Erfahrungen in der Praxis bisher kaum vorhanden sind.

## 2.4 Eignung der Ansätze

In diesem Abschnitt sollen, die in Abschnitt 2.2.2 erkannten Probleme weiter betrachtet werden, um daraus entsprechende Anforderungen an die Anwendung definieren zu können. Diese Anforderungen sollen bei der späteren Bewertung der Technologien als Bewertungskriterien dienen. Am Ende soll eine der hier betrachteten Ansätze ausgewählt werden und später bei der Realisierung verwendet werden.

### 2.4.1 Darstellung eines Anwendungsfalles

In Abschnitt 2.4.3 sollen die einzelnen Ansätze mit einander verglichen werden und auf ihre Eignung in der möglichen Lösung diskutiert werden. Jeder dieser Ansätze, die im Abschnitt 2.3.4 beschrieben wurden besitzt seine Vor- und Nachteile. Diese Vor- und Nachteile ergeben sich aber aus der Betrachtung, wie die Technologien eingesetzt werden sollen.

Aus diesem Grund muss hier ein Vergleich gezogen werden, bei dem alle Ansätze auf einen gleichen Anwendungsfall zum Einsatz kommen, sodass ein objektiver Vergleich zwischen deren Vor- und Nachteilen gezogen werden kann. Daher wird hier als erstes das im

Abschnitt 2.2.1 beschriebene Szenario genauer analysiert. Hierdurch kann das Verhalten des betrachteten Ansatzes besser untersucht werden.

In diesem einfachen Szenario versucht ein Benutzer mithilfe einer Clientsoftware, die sich auf einem PDA befindet, auf die in der unmittelbaren Umgebung befindlichen Ressource, in diesem speziellen Fall einen Server, auf dem ein Dienst residiert, dessen Aufgabe es ist Bahninformationen an mögliche Clients zu liefern, zuzugreifen. In Abbildung 2.14 wird dieses Szenario kurz skizziert.

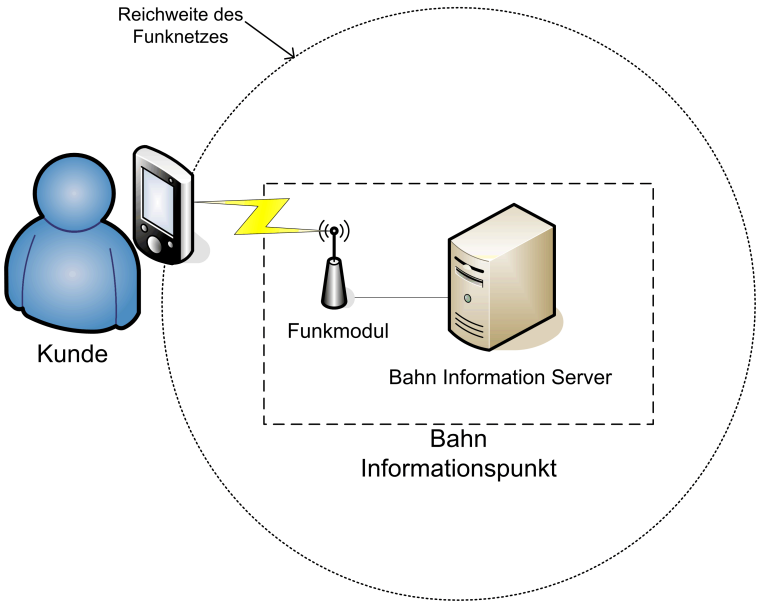


Abbildung 2.14: Grobe Veranschaulichung des Anwendungsszenarios.

Bei diesem Szenario können verschiedene Arbeitsabläufe beobachtet werden. Diese sollten mithilfe der hier entwickelten Software von dem Benutzer einfacher durchgeführt werden. Um diese besser erfassen zu können, wird in der Abbildung 2.15 ein Anwendungsfalldiagramm dargestellt, das die einzelnen Abläufe beinhaltet. Hierdurch wird eine Trennung von der eigentlichen Funktionalität der Kommunikationsmiddleware, also der Verbindungsaufbaufähigkeit und Verbindungsaufrechterhaltung, von den zur Darstellung notwendigen Abläufen besser sichtbar.

Für den Aufbau einer Verbindung zu einer Ressource und dann die Nutzung des dort residierten Dienstes werden oben dargestellte Arbeitsabläufe benötigt. Für den Benutzer existieren zwei Hautprozesse. So kann sich dieser bei einer Ressource und dann einen Dienst anmelden, wodurch der Dienst genutzt werden kann.

Beim Anwendungsfall „Anmelden“ handelt es sich um eine generalisierte Variante der Unteranwendungsfälle. So wird bei dem Unteranwendungsfall „Bei Ressource anmelden“ der Anmeldevorgang zwischen dem mobilen Gerät und der vor Ort befindlichen Ressource verstanden. Am Ende soll eine Verbindung zu der Ressource aufgebaut sein, unter der

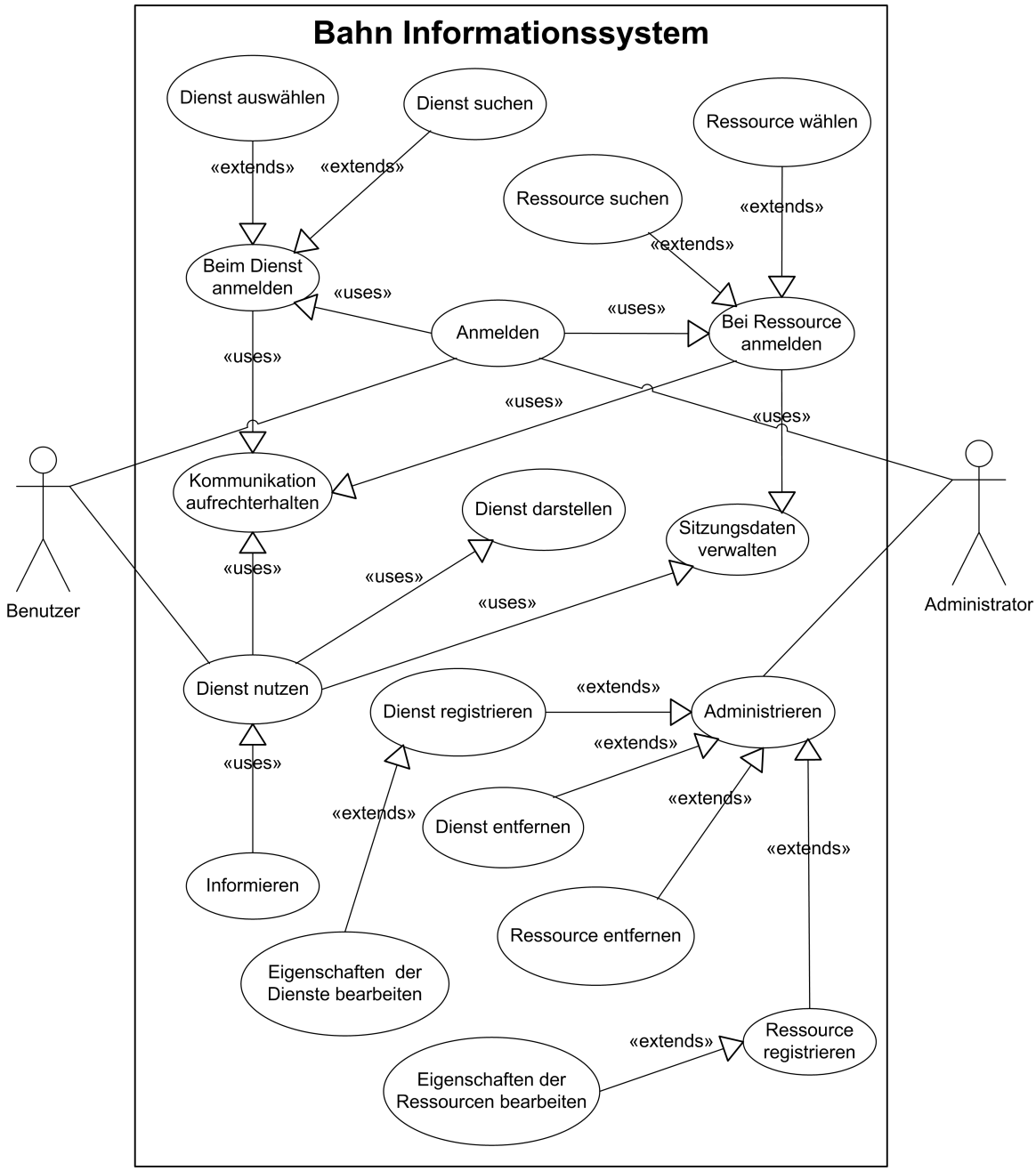


Abbildung 2.15: Anwendungsfalldiagramm für Bahn Informationssystem.

Bedingung, dass die beiden zu kommunizierenden Parteien sich immer noch innerhalb der Funkreichweite befinden. Die Anmeldung wird entweder implizit durch die Erkennung einer Ressource, die sich innerhalb der Reichweite befindet, initiiert oder explizit durch den Benutzer. Dieser Geschäftsprozess wird außerdem durch zwei weitere Prozesse erweitert. Das sind: „Ressource suchen“ und „Ressource wählen“. Hinter dem Arbeitsablauf „Ressource suchen“ verbirgt sich die Funktionalität, die für das Auffinden einer Ressource notwendig ist. Diese hat als Ziel, die Umgebung nach möglichen Ressourcen abzusuchen. Das Suchen wird entweder durch den Benutzer selbst über einen Anmeldewunsch gestartet, oder durch eine Automatik, die in gleichen Zeitabständen die Suche startet. Werden geeignete Ressourcen gefunden, wird dieser Anwendungsfall verlassen und zu dem „Bei Ressource anmelden“ zurückgekehrt, von dem man zu einem anderen „Ressource wählen“ gelangt. In diesem Anwendungsfall wird die Möglichkeit gegeben, zwischen den einzelnen Ressourcen wählen zu können. Diese werden hier durch unterschiedlichste Güteeigenschaften unterschiedlich dargestellt. Am Ende sollen dann die Ressourcen, die den gestellten Eigenschaften entsprechen, automatisch ausgewählt werden. Diese Auswahl wird automatisch von dem Clientprogramm durchgeführt. Das Auswahlverfahren hängt von den Benutzeranforderungen ab und kann jederzeit geändert werden.

Nach der Auswahl wird wieder zu dem Anwendungsfall „Bei Ressource anmelden“ verzweigt, wo dann versucht wird, mit der Ressource eine Verbindung aufzubauen. Dabei können Störungen auftreten, die erkannt und entsprechend behandelt werden müssen. Hierfür ist der Anwendungsfall „Kommunikation aufrechterhalten“ zuständig. Dieses wacht über die zurzeit laufenden Verbindungen, solange diese noch von Bedeutung für den Benutzer und das System sind. Wurde eine Verbindung unterbrochen, obwohl diese immer noch für den Benutzer relevant war, werden hierfür notwendige Maßnahmen getroffen, diese wieder aufzubauen. Dabei werden die damit verbunden Schritte nach Möglichkeit für den Benutzer transparent durchgeführt. Benutzer und somit die Clientanwendung sollten von der Verbindungsproblematik weitgehend entkoppelt sein.

Kann nach dem Verlust einer Verbindung diese nicht wieder aufgebaut werden, so wird zu dem Prozess „Sitzungsdaten verwalten“ übergegangen. Mit diesem sollen alle wichtigen Sitzungsdaten im Client, sowie im Server, gespeichert werden, sodass keine Inkonsistenzen entstehen können. Die Speicherung der Daten erfolgt aber auch aus dem Grund, um später bei einer möglichen Verbindung mit dem gleichen Dienst den Arbeitsvorgang fortsetzen zu können. Im Großen und Ganzen ist dieser Anwendungsfall also für das Sichern und Wiederherstellen der nicht vollständig zu Ende durchgeführten Sitzung zuständig.

Wird eine stabile Verbindung zu einer Ressource aufgebaut, so wird der Benutzer die Möglichkeit haben, sich mit den Diensten, die auf diesem Server residieren, zu verbinden. Hierfür erforderliche Funktionalität wird in dem Geschäftsprozess „Beim Dienst anmelden“ zusammengefasst. Als Vorbedingung für einen Verbindungsaufbau wird ein verfügbarer Dienst vorausgesetzt. Ist ein solcher Dienst vorhanden, so wird versucht, mit ihm eine Verbindung aufzubauen.

Um einen verfügbaren Dienst zu finden, wird zum Arbeitsablauf „Dienst suchen“ verzweigt. In diesem wird eine einfache Suche ermöglicht, bei der verschiedene Eigenschaften der Dienste berücksichtigt werden können. Eine Dienstsuche kann aber nur dann durchgeführt werden, wenn eine Verbindung zu einer Ressource[6] steht. Ist eine Verbindung vorhanden, so kann der Benutzer explizit nach Diensten, die seinen Sucheingaben entsprechen, suchen oder per Voreinstellung eine Suche automatisch von der Clientsoftware durchführen lassen. Werden entsprechende Dienste gefunden, wird in dem Arbeitsablauf „Dienst auswählen“ ein Dienst von dem Benutzer ausgewählt. Dieser Anwendungsfall unterstützt den Benutzer bei der Auswahl des für ihn geeignetsten Dienstes. Wird ein Dienst gewählt, so kann eine Verbindung zu diesem aufgebaut werden. Hierfür notwendige Funktionalität verbirgt sicht hinter dem Geschäftsprozess „Beim Dienst anmelden“.

Hat schließlich der Client eine Verbindung zu einem Dienst aufgebaut, so muss diese ebenfalls über die Dauer der Session überwacht werden, was in Geschäftsprozess „Kommunikation aufrechterhalten“ passiert. Bleibt die Verbindung zu dem Dienst bestehen, so ist der Benutzer in der Lage, diesen zu nutzen. Die Nutzung eines Dienstes wird in dem Geschäftsprozess „Dienst nutzen“ abgehandelt. Um eine Interaktion zwischen dem Dienst und dem Benutzer ermöglichen zu können, ist eine grafische Darstellung des Dienstes notwendig. Die hierfür notwendige GUI[7] und alle damit verbundenen Tätigkeiten werden in dem Prozess „Dienst darstellen“ behandelt.

Ist auch das Darstellungsproblem gelöst, so kann der Benutzer zu dem Anwendungsfall „Dienst nutzen“ übergehen. In diesem ist der Zugriff auf notwendige Informationen der Bahn enthalten. Der Zugriff wird in dem Unterprozess „Informieren“ behandelt. Hier werden die Suchbedingungen eingegeben und die geforderten Informationen ausgegeben.

Außer der Nutzung der Ressourcen und Dienste müssen diese ebenfalls administriert werden. Hierfür muss sich ein Administrator an einer Ressource anmelden können, was durch den bereits genannten Anwendungsfall „Anmelden“ behandelt wird. Ist ein Administrator angemeldet, stehen ihm verschiedene Möglichkeiten der Administration zur Verfügung. So kann dieser einmal eine Ressource bzw. einen Dienst administrieren. Beide sind zu differenzieren, besitzen aber ähnliche Unterprozesse.

### 2.4.2 Bewertungskriterien für die Ansätze

Jeder Ansatz hat seine Vor- und Nachteile. So tauchen in diesem Zusammenhang verschiedene Fragen auf, wie z.B. welche Vorteile von großer Bedeutung sind, welche nicht? Welche Nachteile kann man in Kauf nehmen, welche nicht. Bei den negativen Merkmalen ist analog vorzugehen. Um eine objektive Bewertung der Ansätze machen zu können, werden Bewertungskriterien benötigt nach denen die einzelnen im Abschnitt 2.3.4 vorgestellten Ansätze bewertet werden können. Im Folgenden werden die einzelnen Gesichtspunkte, nach dem

[6] Gemeint ist hier ein Server, auf dem die Multimedia Dienste zu finden sind.

[7] Graphic User Interface

die Bewertung erfolgen wird, genannt und beschrieben. Diese sind aber auch automatisch die Anforderungen die der Multimedia Dienst und der Client, der auf diesen Dienst zugreift, an die mobile Multimedia Middleware stellen.

**Flexibilität**

Hier ist ein modulares System gemeint, welches die Fähigkeit besitzt mit Änderungen umzugehen. Im Genaueren, ist in diesem Zusammenhang die Unwissenheit über die Methoden der angesprochenen Dienste, also den Ausführungskontext, gemeint. Durch diese Gegebenheit, kein Wissen über den Kontext voraussetzen zu müssen, wird der größte Teil der Mobilität begünstigt. Hierdurch wird die Anpassung an die Umwelt vereinfacht, was eigentlich bei der Bereistellung der Mobilität als einer der Hauptaspekte zu sehen ist. So sollte eine Anwendung ein Minimum an Wissen über den unbekannten Dienst vor dem Verbinden mit diesem voraussetzen. Alle wichtigen Informationen über die aktuelle Umgebung sollten mithilfe der in dieser Arbeit zu entwickelnden Software abgerufen werden können. Das erforderte Wissen sollte auf dem Server, auf dem sich die Dienste befinden, zentralisiert werden. Hierdurch wird jede Änderung für alle sichtbar.

**Interoperabilität, Portabilität und Offenheit**

Gemeint ist hier die Zusammenarbeit zwischen den, eventuell auf verschiedenen Plattformen residierten, Diensten und Clients. Diese sollten unabhängig von dem Betriebssystem, miteinander koexistieren und zusammen arbeiten können. Es macht wenig Sinn ein mobiles Programm zu entwickeln, welches sich physikalisch bewegen kann, aber nicht auf andere Anwendungen zugreifen kann, weil diese auf anderen Plattformen laufen. Des Weiteren sollte das Design der mobilen Multimedia Middleware die darauf laufende Anwendung so weit wie möglich von dieser entkoppeln, sodass diese mit so gut wie keiner Änderung der Software, auf einem anderen verteilten System lauffähig ist. Außerdem muss der Aspekt der Offenheit berücksichtigt werden, damit eine Erweiterung des Systems leicht zu erzielen ist.

**Einfachheit**

Damit ist die Komplexität der Middleware gemeint, die so weit wie es nur möglich ist, klein gehalten werden soll. Es nutzt keinem sehr viel, wenn eine Middleware zu komplex und somit zu unhandlich ist. Da die Middleware zur Kommunikation zwischen Server und Client benutzt wird, ist es empfehlenswert, die eigentliche Implementierung der Kommunikation so weit wie möglich von der eigentlichen Implementierung der Dienste und Clients zu entkoppeln. Dadurch wird die Einfachheit bei der Client- und Dienstentwicklung unterstützt.

**Performance**

Die Performance ist ebenfalls ein wichtiger Aspekt. Die spätere Lösung muss im Stande sein, einer Antwort innerhalb eine akzeptablen Zeit liefern zu können. Auf der anderen Seite darf es aber für die Bearbeitung der gestellten Aufgaben nicht die gesamte Rechenleistung, die zur Verfügung steht, aufbrauchen. Es ist zu beachten, dass während der Kommunikation, parallel auch noch andere Anwendungen laufen sollen. Dabei ist zu beachten, dass dieser Aspekt zum jetzigen Zeitpunkt von Bedeutung ist. Die Rechenleistung der hier betrachteten Geräte ist nicht in ähnlicher Größenordnung, wie das derzeitig bei einem PC vorzufinden ist. Man kann aber davon ausgehen, dass dies in naher Zukunft Zeit aufgeholt werden kann und somit dem Aspekt der Performance, nicht mehr solche große Bedeutung zugeschrieben wird.

**Pflege und Wartbarkeit**

Mit dem Betriebsbeginn der Software beginnt seine Wartung. Bei der Wartung sollte die Software weiter stabilisiert werden, also die zu diesem Zeitpunk gefundenen Fehler sollten korrigiert werden können. Mit anderen Worten, soll eine Möglichkeit gegeben sein, später zum Vorschein kommenden Designschwächen, mit relativ wenig Arbeitsaufwand zu beseitigen, ohne dabei eine Angleichung des gesamten Systems machen zu müssen. Außerdem sollte es möglich sein die Software weiter optimieren zu können, um Leistungsverbesserung zu erreichen. Die mobile Multimedia Middleware sollte so weit es geht änderbar, analysierbar und modifizierbar sein, damit die Wartbarkeit unterstützt werden kann. Durch diese Eigenschaft werden der Aufwand der Diagnostizierung der Fehler und ihre Beseitigung verkürzt.

**Rekonfigurierbarkeit**

Im Stande sein so weit es geht eine Änderung des Dienstes zu machen, ohne dabei die Implementierung des Clients anfassen zu müssen. Gemeint sind die Veränderung des Funktionsumfangs eines Dienstes und die Angleichung des Clients ohne Neuübersetzung. So soll hierbei eine Entkopplung der beiden kommunizierten Parteien vorgenommen werden, sodass hierdurch die Änderung der einen Seite die Änderung bzw. Angleichung des anderen nicht notwendig macht.

**Stabilität**

Ein vollständiges System (komplexes Programm) kann nicht stabil entwickelt werden, aber einige Teile können als stabil betrachtet werden. Aus diesem Grund sollte die Software als eine stabile Komponente entwickelt werden, da das als Herz der Architektur anzusehen ist. Diese soll die Fähigkeit besitzen aus einem unerwarteten Zustand in einen definierten übergehen zu können, sodass das System weiter betriebsbereit ist.

### Ortstransparenz

Ortstransparenz bedeutet im diesem Fall, dass der Benutzer keine Aussage über den physikalischen Aufenthaltsort der Ressource in dem System treffen kann. Die Nutzbarkeit der Dienste sollte unabhängig vom Ausführungskontext des Dienstes sein, d.h., es sollte für den Benutzer keine Rolle spielen, ob sich der Dienst auf dem lokalen Rechner, in einem verteilten System oder in einem entfernten Kommunikationsknoten befindet. Der Dienst verhält sich immer gleich. Hierdurch entfällt bei der Cliententwicklung der Teil der Auffindung der Ressource, was die Entwicklungszeit verkürzt, die Leistung des Clients steigert und den Client einfach hält. Durch die Einhaltung dieser Forderung wird die physikalische Fortbewegung des Dienstes sowie des Clients ebenfalls begünstigt, also auch die Mobilität, die das spontane Verbinden mit enthält. Hierdurch wird ein Client im Stande sein bei Bewegung von Standort zu Standort ohne zeitweilige Verbindungsunterbrechung, weiter den Dienst benutzen zu können.

### Spontane Verbindungsaufbaufähigkeit

Gelangt ein mobiles Gerät in eine unmittelbare Umgebung eines Dienstes, wird dieses transparent konfiguriert und dann eine Verbindung mit dem Dienst aufgebaut. Die Forderung an dieser Stelle ist, dass der Benutzer so wenige wie nur möglich, administrative Aufgaben leisten sollte, um eine Verbindung mit dem Server aufbauen zu können. Unter Spontanität wird an dieser Stelle eine Verbindungsbereitschaft verstanden, sodass der Benutzer nicht damit beschäftigt ist Dienste zu suchen und sich mit denen zu verbinden. Dieser sollte sich viel mehr mit der Benutzung der zu Verfügung stehenden Dienste beschäftigen. Die Software sollte im Stande sein, zwischen den Beiden eine Verbindung sofort, bei der ersten Gelegenheit aufzubauen. Dadurch soll eine bequeme Verbindung und Integration unterstützt werden. Mit der Spontanität werden aber noch weitere Probleme behoben. Da diese im Umfeld der Kommunikation eingesetzt wird, kann somit diese vollständig nach außen verborgen werden. Hierdurch wird die Diensteimplementierung zusätzlich von dieser Problematik entlastet.

### Eingeschränkte Verbindungsaufrechterhaltung

Wird bei der Kommunikation die Verbindung unterbrochen, soll dies zu keinem Absturz des Clients sowie des Dienstes führen. Die beiden hier kommunizierenden Seiten sollen im Stande sein, sich von solcher abrupten Verbindungsunterbrechung zu erholen. Im Genaueren heißt das, dass die Kommunikationsaufrechterhaltung so weit von den beiden kommunizierenden Partnern entkoppelt sein soll, sodass diese für die transparent im Hintergrund behandelt wird. Die mobile Multimedia Middleware soll die Tatsache des unvorhersehbaren Kommunikationsverlustes handhaben können, sodass die Client- und Serverimplementierung davon befreit sind.

#### Generalisierte Erkennung der Dienste

Es soll für den Benutzer möglich sein, nach den in der Umgebung stationären Diensten, mit der geeigneten Dienstgüte und Eigenschaft, zu suchen und diese auszuwählen. Die Software sollte somit den Benutzer bei der Suche nach einem geeigneten Dienst unterstützen, bei der dieser für ihn geeignete und geforderte Suchvorgaben definieren kann. Die Dienstbeschreibung sollte standardisiert werden, sodass eine generalisierte Suche begünstigt wird.

### 2.4.3 Die Ansätze diskutieren

In diesem Abschnitt sollen die im Abschnitt 2.3.4 vorgestellten Ansätze bewertet und einer davon ausgewählt werden. Dieser wird dann in dem hier erarbeiteten Lösungsansatz verwendet. Die Bewertung wird mithilfe der im Abschnitt 2.4.2 vorgestellten Kriterien durchgeführt.

#### Web Services

Die Web Services Technologie führt eine relativ neue und innovative Vorgehensweise in dem Umfeld der verteilten Systeme ein. Diese verfolgt das Prinzip, wie alle anderen Vorgänger, die mit dem *Component Computing Model* vorgestellt wurden. So kann in diesem Zusammenhang DCOM, welches eng mit COM verknüpft ist, sowie CORBA, als Vorgänger genannt werden.

Ein gemeinsames Manko, welches die Vorgänger besitzen, ist die Voraussetzung, dass beide Kommunikationspartner auf Plattformen laufen, auf denen notwendig COM und DCOM, bzw. CORBA vorhanden sein muss. Dieses engt die Interoperabilität ein. Um dieses Hindernis zu umgehen, müssen verteilte Systeme von einer bestimmten Plattform entkoppelt werden. Diese Eigenschaft wird in (Westphal, 2001) folgendermaßen definiert: „Der technologische Aufwand, um eine ferne Komponente aufrufen zu können, sollte möglichst klein sein, d.h. eine dezentralisierte Technologie möglichst leichtgewichtig.“ Dieses wird eben aus Sicht der Web Services gewährleistet. Durch ihren Aufbau, welcher hauptsächlich auf der XML-Technologie basiert, werden diese somit vollständig sprach- und plattformunabhängig, wodurch diese einen Schritt gegenüber den Vorgängern voraus sind. Durch diese Vorteile werden bei Web Services keine Voraussetzungen an die Laufzeitumgebung, sowie an das Objektmodel gestellt. Diese werden mit Web Services abstrahiert, sodass keine Notwendigkeit besteht, dass die aufzurufenden Softwarekomponenten dieselbe Systemarchitektur verwenden müssen.

XML stellt aber noch einen weiteren Vorteil bereit. Als Transportmedium für die Daten, die ausgetauscht werden, eliminiert XML alle Netzwerkbindungen eines Protokolls. Die Vorgängersysteme hatten aber noch mit anderen Problemen zu kämpfen. So waren bei diesen die beiden Kommunikationsparteien stark aneinander gekoppelt. Hierdurch wurde die Änderbarkeit stark behindert. Die Web Services Technologie stellt ein lose gekoppeltes ver-

teiltes System dar. Dies bedeutet, dass eine Änderung einer der Kommunikationspartner keine Auswirkung auf die Verfügbarkeit des Anderen hat, wodurch die Änderbarkeit in größter Masse gewährleistet wird. Die Web Services entkoppeln die Softwarekomponenten aber noch weiter. So können diese, durch die zur Verfügung stehende asynchrone Ausführung, die Kommunikationspartner noch weiter voneinander entkoppeln.

Heutzutage ist es gängig, dass Unternehmen ihre Firmennetze durch Firewalls gegenüber Internet abschotten. Dieses stellt ebenfalls bei den Vorgänger Systemen ein Problem dar. Bei Einsatz bestimmter Techniken, bei dem Internetports benötigt werden, machen Firewalls diese nämlich fast unmöglich, da eine Firewall diese nicht zulässt. Web Services umgehen diese Problematik, indem sie für die Kommunikation HTTP einsetzen. Durch diese eben genannten Vorteile wird das Hauptziel, das durch die Web Services verfolgt wird, deutlich. Dieses ist die nahtlose Interoperabilität über heterogene Systeme, Plattformen, Anwendungen und Programmiersprachen hinweg. Diese Interoperabilität ist aber nur dann erlangt, wenn das gesamte Konzept sauber und einheitlich durchzogen wird, da die Web Services die Interoperabilität nicht garantieren. Durch diese Vorteile eröffnen sich neue Möglichkeiten, die durch die bisherigen Technologien nicht gegeben waren. Alte sowie neue Softwarekomponenten können ohne eine Bindung an die andere Kommunikationsseite im Netzwerk verfügbar gemacht werden. Die Wiederverwendbarkeit, Änderbarkeit, Unabhängigkeit spielen bei Web Services die Kernaspekte.

So schön wie das alles auch klinkt, besitzen Web Services aber auch einige Nachteile. Diese sind einmal durch die Unausgereiftheit gegeben, sodass die Innovativität sich auch im negativen Sinne hier kenntlich macht. Bei den Web Services werden verschiedene Technologien benutzt, die teilweise immer noch nicht spezifiziert sind, bzw. in unterschiedlichsten Versionen vorhanden sind. So entstehen hierbei, bei der Verwendung dieser in unterschiedlichsten Versionen, weitere Kompatibilitätsprobleme, da diese nicht zwingend abwärts kompatibel sein müssen. Außer den verwendeten Hilfstechnologien werden ebenfalls die Web Services weiter entwickelt, was durch fehlende Versionisierung wiederum zu Kompatibilitätsproblemen führen kann. Diese Versionisierungsprobleme nehmen den Web Services einen deutlichen Teil der Interoperabilität weg.

Außer den eben genannten Problemen verlieren die Web Services an anderen Stellen weiter an Interoperabilität. So wird durch eine Fehlinterpretation der Spezifikationen, welche in sich selbst manchmal nicht eindeutig ist und an einigen Stellen selbst widerspricht, die Kommunikation gestört. Probleme entstehen ebenfalls durch die noch nicht spezifizierten Anforderungen, die an Web Services gestellt werden. Hierdurch werden diese durch proprietäre Erweiterungen ergänzt, was die Inkompatibilität untereinander erhöht.

Die Web Services werden als eine offene Technologie gepriesen. Diese Offenheit verursacht aber auch neue Probleme. So können für die Auffindung der Dienste unterschiedliche Registrierungsmechanismen und Dienstdefinitionen verwendet werden. Für die Dienstdefinitionen werden verschiedene Werkzeuge benutzt, welche die Generierung der komplizierten WSDL vor dem Entwickler verbergen, was wiederum von verschiedenen Herstellern unter-

schiedlich realisiert werden kann.

Ein weiteres Problem entsteht z.B. bei der Verwendung der WSDL, welches wiederum auf die Offenheit zurück zuschließen ist. So gibt es unterschiedliche Meinungen, was die Nützlichkeit der WSDL angeht. So meint eine Gruppe, dass diese nur ein Hindernis zu Erreichung der Interoperabilität darstellt, das andere Lager behauptet aber, dass diese die Interoperabilität begünstigen. So müssen Dienste nicht immer die WSDL enthalten, was im eingeschränkten Kontext akzeptabel ist, was aber in dem gesamten Kontext der Web Services zu Problemen führen kann. Um diese genannten Probleme einigermaßen umgehen zu können, ist eine vorherige Kommunikation erforderlich. Dabei findet eine Einigung zwischen den beiden Kommunikationspartnern statt, um eine fehlerfreie Kommunikation gewährleisten zu können. Ein solches Vorgehen widerspricht aber dem, was Web Services Technologie bereitstellen soll.

Außer den bereits erkannten Problemen können noch weitere hinzukommen. Diese können möglicherweise bei der Weiterentwicklung der Spezifikationen entstehen, die die fehlenden Aspekte wie Zuverlässigkeit, Sicherheit und Dienstqualität spezifizieren werden. Es ist zu hoffen, dass bei der Spezifizierung neuer Anforderungen, die Gremien nicht die gleichen Fehler begehen werden, wie das bis jetzt der Fall war. Gemeint ist die fehlende Koordination untereinander, wodurch sich widersprechende, konkurrierende und sich überschneidende Standards entstanden sind, was die Interoperabilität der Web Services eher erschwerte.

Allgemein lässt sich sagen, dass die Web Services einen interessanten Weg gehen, wodurch dem verteilten System neue Eigenschaften verliehen werden. Dieser Weg ist aber noch weiter fortzuführen, da diese Ansätze sich immer noch im Entwicklungsstadium befinden. Es gibt noch an vielen Stellen Mängel, welche beseitigt werden müssen.

#### J2EE vs. .NET

Wie zuvor schon erwähnt, ähneln sich die beiden Plattformen, zumindest was die Unterstützung der Web Services Technologie angeht. Bei der weiteren Betrachtung der beiden Plattformen kommen jedoch deren Unterschiede zum Vorschein. Diese sollten hier näher betrachtet werden.

Wenn man die Offenheit betrachtet, so ist J2EE ein Satz offener Standards, aber kein Produkt. Dieses sieht bei .NET anders aus. Dabei handelt es sich um eine Produktfamilie, bei der ein Teil auf Standards basiert, während ein anderer Teil die Standards auf proprietäre Weise erweitert. In dieser Hinsicht ist also J2EE eine mehr offene Plattform als .NET, was für Web Technologien eine wichtige, wenn nicht sogar fundamentale Forderung ist.

In diesem Zusammenhang kann ein weiterer Unterschied genannt werden. So konzentriert sich J2EE auf die Anwendungsportabilität und Integration von Plattformen, die Java unterstützen. .NET geht in diesem Modell eine Abstraktionsebene höher und behauptet, die Integration von Anwendungen über Plattformen mittels XML zu erreichen. Dieser Aspekt wird aber inzwischen auch von J2EE 1.4 berücksichtigt, da diese Version die benötigten Teile der

Java *Web Services Deverloper Pack* (WSDP) enthält, durch welche die Web Services weitgehend unterstützt werden.

Weiter zu erwähnen ist, dass .NET integrierte Unterstützung zum Entwickeln und Debuggen XML basierter Web Services enthält. Diese umfasst eine große Werkzeugspalette die mit der .NET Plattform mitgeliefert werden. Bei J2EE müssen solche von den entsprechenden J2EE-Herstellern angeboten werden. Außerdem stellt die .NET Plattform eine Laufzeitumgebung für SOAP und UDDI als native .NET Protokolle bereit.

Unterschiede sind ebenfalls bei der Integration von Anwendungen und Backends zu finden. Bei der Integration hat sich Microsoft viele Gedanken gemacht. So stellt .NET verschiedene Möglichkeiten dar, mit denen verschiedene Integrationsaufgaben gelöst werden können. In diesem Zusammenhang tauchen Technologien auf wie: *Host Integration Server*, *COM Transaction Integrator*, *Microsoft Message Queue* und *BizTalk Server*, die hier nicht weiter beschrieben werden. Außerdem wird auf der .NET Plattform die Möglichkeit für Geschäftsprozessverwaltung und E-Commerce geboten. Bei J2EE kann diese Fähigkeit angeboten werden, ist aber kein Bestandteil des Standards.

Sun setzt groß auf Offenheit, sodass J2EE auf allen gängigen Plattformen verfügbar ist. Bei Microsoft sieht die Strategie etwas anders aus, was auch durch den kommerziellen Vertrieb des .NET-Frameworks gezeigt wird. Außerdem ist dieser nur auf dem Microsoft Plattformen verfügbar. Hierdurch ergibt sich die eine Abhängigkeit von dem Betriebssystem, was bei offenen Systemen nicht wünschenswert ist. In dieser Hinsicht unterscheiden sich also beide Plattformen in der Portierbarkeit. So können bei J2EE alle Anwendungen auf alle Plattformen portiert werden, bei .NET nicht.

Ein Unterschied ist ebenfalls in dem Konzept der virtuellen Maschinen festzustellen. Diese unterschieden sich in der Hinsicht, dass die JVM an die Sprache Java gekoppelt ist. Microsoft hat das sprachspezifische Element aus dem Zwischencode entfernt, sodass theoretisch jede Programmiersprache in diese so genannten *Microsoft Intermediate Language* (MSIL) übersetzt werden kann. Die damit arbeitende *Common Language Runtime* (CLR) unterscheidet dann nicht mehr, in welche Programmiersprache das Programm geschrieben ist. Diese übersetzt dann zur Laufzeit (Just In Time) diesen Code in den native Code und führt in aus.

Was die Verfügbarkeit der Hilfswerkzeuge angeht, unterscheiden sich beide Plattformen kaum voneinander. Bei der Javaplattform wird für Buildprozesse das bewährte, freie ANT Werkzeug, für Modultest das JUnit, sowie für Dokumentation das JavaDoc zur Verfügung gestellt. Bei der .NET Plattform existieren vergleichbare Werkzeuge wie NANT, NUnit, NDoc. Diese werden ebenfalls als Freeware zur Verfügung gestellt. Web Services werden von beiden Plattformen unterstütz. Dabei steht im Vordergrund die Integrationsfähigkeit, also die Fähigkeit mit anderen Anwendungen Datenaustausch betreiben zu können.

Der Unterschied der zwischen den beiden Plattformen bei der Unterstützung der WS ist der, dass .NET speziell für die Unterstützung von WS und deren Hilfstechnologien (UDDI, WSDL, SOAP) entwickelt wurden und somit besser in .NET integriert sind.

Web Services und J2EE haben beide die gleiche Hauptanwendungsdomäne, das Internet. Diese sollen die Entwicklung der Internetanwendungen erleichtern. Eine typische Internetanwendung wird logisch in vier Schichten zerlegt. Diese sind: Client, Präsentationsschicht, Logikschicht, Datenzugriffsschicht. In der weiteren Betrachtung werden die Besonderheiten der beiden Technologien in der Logikschicht, Präsentationsschicht und Datenzugriffsschicht hervorgehoben. So verwendet J2EE die Servlets oder JSP Seiten für Darstellungszwecke, was bei .NET dafür die ASP.NET Technik vorsieht. Diese stellt eine Weiterentwicklung der ASP-Technik dar. Auf der Logikschicht kommen bei J2EE die Session Beans zum Einsatz, welche eine Teilmenge der EJB darstellen. Hierdurch ist ein lokaler sowie ein entfernter Zugriff möglich, wobei die Position für den Benutzer transparent ist (location transparency). Im Gegensatz dazu steht bei .NET keine Unterstützung für die Logikimplementierung zur Verfügung.

Wie J2EE erlaubt auch .NET einen lokalen sowie entfernten Zugriff auf Objekte. Dieser hat aber einen gravierenden Nachteil, die Location Transparency ist nicht gewährleistet. Auf der Datenzugriffsschicht stellt J2EE ebenfalls eine Untermenge der EJB, die Entity Beans bereit. Diese sorgen für einen persistenten Datenzugriff, welches über einen Applikationsserver automatisch durchgeführt werden kann. Bereitgestellt wird auch eine zweite Variante, in der der Entwickler manuell den Datenzugriff programmiert. Bei .NET kommt an dieser Stelle die ADO .NET Technik. Diese ist stark in der .NET Plattform integriert und mit den darin befindlichen Technologien wie Verarbeitung und Kommunikation von Daten im XML-Format, was einen großen Vorteil darstellt. Werden große Mengen von Daten über ASP.NET und HTML oder WS in Verbindung mit SQL Server zur Verfügung gestellt, stellt ADO .NET eine gute Lösung dar.

Die Stärken der J2EE in diesem Bereich liegen im Allgemeinen in der Flexibilität und Skalierbarkeit dieser Architektur, die in der Persistenzschicht einen stark objektorientierten Ansatz verfolgt. Die flexible Spezifikation und hiermit verbundene effektive Trennung von Anwendungsschnittstellen und Datenbank, die bei J2EE vorzufinden ist, eignet sich dieser Ansatz speziell für sehr dynamische Informationssysteme, bei den sich schnell ändernde Daten befinden. Bei dieser Betrachtung wird ersichtlich, dass J2EE mit Hilfe von EJB serverseitig ein Komponentenmodell vorgibt. Dieses ist bei .NET nicht der Fall. Hinzu kommt noch, dass bei J2EE Aufgaben die von einem Applikationsserver im Verantwortungsbereich stehen, bei .NET von Betriebssystem übernommen werden, was eine weitere Betriebssystemabhängigkeit darstellt. Wie im vorherigen Abschnitt erwähnt wurde, stellt die .NET Plattform eine Möglichkeit dar, Anwendungen in mehreren Programmiersprachen zu entwickeln, welche dann auf einer gemeinsamen virtuellen Maschine laufen und miteinander agieren können. Dieses Konzept wird bei J2EE nicht unterstützt.

### CORBA

**Statische Methodenaufruf über CORBA** Das statische Interface wird direkt in der Stellvertreterform (Stub) von dem IDL-Compiler generiert. Es ist ideal für Programme, welche zur Übersetzungszeit die Einzelheiten der Methoden, welche aufgerufen werden können, kennen. Das statische Stub-Interface wird zur Übersetzungszeit gebunden. In diesem Abschnitt sollen Vor- und Nachteile diskutiert werden, die sich bei dem Einsatz vom reinen statischen Methodenaufruf, bei der Kommunikation zwischen dem Dienst und Client, ergeben. Der statische Methodenaufruf liefert die folgenden Vorteile gegenüber dem dynamischen Methodenaufruf:

**Performance** Durch die zur Übersetzungszeit bekannte Signatur der zu implementierenden Methoden des Objekts und daraus generierten Stubs, ergibt sich wohl der wichtigste Vorteil bei dem statischen Methodenaufruf, gemeint ist hier die Performance. Diese entsteht aus dem Grund, weil kein zusätzlicher Aufwand bei dem Auffinden der richtigen Methoden betrieben werden muss. Der Performanceunterschied zum dynamischen Methodenaufruf ist erheblich, sodass dieser Ansatz bei zeitkritischen Anforderungen vorgezogen werden sollte.

**Typüberprüfung durch den Compiler** Dadurch, dass zur Übersetzungszeit alle Methoden bekannt sein müssen, können sich weniger Fehler einschleichen, da der Compiler sonst sofort Alarm schlagen würde. Der Compiler liefert mehr robuste Typüberprüfung. Hinzu kommt noch die richtige Benutzung der Methoden, da diese vorher in einen Standard genügend beschrieben werden und dadurch sich keine Zweideutigkeiten bei der Interpretation der Methodenbezeichnungen ergeben können, wodurch die Entwicklungszeit der Clients verkürzt wird. Hinzu kommt noch die enorme Kompatibilität auf der Clientseite, welche durch die in dem Standard fest definierte, sich nicht ändernde Schnittstelle entsteht. Jeder Client kann mit jedem Dienst kollaborieren, soweit der Standard von beiden Seiten eingehalten wird.

**Standardisierte Schnittstelle** Ebenfalls ist als Vorteil die mögliche Verlagerung der Verarbeitung der Clientanfragen auf die Serverseite, wo sich die Dienstimplementierung befindet, zu sehen. Gemeint ist hiermit eine soweit standardisierte Schnittstelle, bei der die Methoden bekannt sind. Dadurch wird der Client von der Suche nach möglichen Dienstmethoden fast vollständig entlastet, da die möglichen Dienstmethoden zur Entwicklungszeit bereits bekannt sind.

**Einfach zu Programmieren** Es ist einfacher zu programmieren. Man ruft die entfernte Methode durch einfachen Aufruf deren Signatur, die sich in dem Stellvertreter (Stub) befindet, und übermittelt deren Parameter. Diese Vorgehensweise lässt sich als eine sehr natürliche Form der Programmierung bezeichnen, was die Entwicklung von Clients unterstützt.

**Selbsterklärend** Statischer Methodenaufruf ist selbsterklärend. Durch diese Eigenschaft wird der Programmkode übersichtlicher und lesbarer, wodurch eine spätere Wartung und Pflege begünstigt wird.

Wie viele andere Ansätze in der Informatik, hat auch dieser nicht nur Vorteile, sondern ebenfalls Nachteile. Das sind:

**Flexibilitätsverlust** Für die hohe Performance zahlt man mit dem Verlust der Flexibilität. Man kann mithilfe des statischen Methodenaufrufs keinen bereits existierenden Client bzw. Dienst nutzen, ohne ihn vorher auf die Schnittstelle angepasst zu haben. Hierdurch wird die Evolution eines Dienstes erheblich verhindert. Betrachtet man in diesem Zusammenhang die Mobilität des Clients, die ebenfalls die Fähigkeit, sich an die Umgebung anzupassen beinhaltet, ist dies hier nicht gewährleistet.

**Anpassbarkeit gestört** Aus der Sicht der Kosten ist dieser starre Ansatz bei deren Wartung und Pflege teurer als der dynamische Ansatz. Durch eine Einführung eines neuen Dienstes, bei dem die Schnittstelle nicht mehr dem alten Standard entspricht, werden hier alle Clients unbrauchbar, was eine Anschaffung einer angeglichenen Version erfordert. Hinzu kommt noch, dass hierdurch in gewisser Weise die Portabilität eingeschränkt wird. Der Client kann nicht von einer alten in eine neue Umgebung bewegt werden, in der eine unbekannte Schnittstelle unterstützt wird, wodurch die Mobilität gestört ist.

**Erschwerte Änderbarkeit** Wird am Anfang in der Planungsphase eine fehlerhafte Schnittstelle vereinbart und die Designfehler erst später in Betrieb erkannt, kann es sich schon aus Kostengründen eine Beseitigung der Fehler nicht mehr lohnen. Die Fehler werden weiter mitgeschleppt.

**Verhinderung der Erweiterbarkeit** Jede Änderung der Schnittstelle, sei sie noch so klein, verursacht meistens die Anpassung der existierenden Clients beziehungsweise Dienste. Die Erweiterbarkeit wird hierdurch zum größten Teil verhindert.

Dieser Ansatz kann zum Beispiel auf ein System angewendet werden, bei dem die Performance der wichtigste Aspekt ist. Hierdurch erreicht man den größten Durchsatz an Anfragen innerhalb einer akzeptablen Antwortzeit. Also kann man sagen, dass dieser Ansatz sich für eine Echtzeitanforderung am besten eignet. Da bei diesem Ansatz die Flexibilität fast gar nicht unterstützt wird, wird das System sehr kontextspezifisch d.h., für jeden Kontext wird eine andere Schnittstelle entwickelt. Als eine mögliche Erweiterung dieses Konzepts, um die Flexibilität unterstützen zu können, ist die Erweiterung der Schnittstelle um Methoden, die als Aufgabe haben, den Kontext zu erkennen und somit das Verhalten der Schnittstelle darauf anzupassen. Diese Methoden bauen weiter auf dem statischen Ansatz auf, sodass deren Vorteile weiter gegeben werden.

Als Beispiel für so ein System ist eine einfache Verbindung zwischen einem Client und einem kontextspezifischen Dienst. Dieser liefert in kurzer Zeit die geforderten Informationen, welche ebenfalls kontextspezifisch sind. So eine Schnittstelle kann z.B. für die Übermittlung von Daten zwischen Dienst und Client konzipiert sein. Diese Daten werden dann im Dienst ausgewertet, bzw. berechnet, damit die Ergebnisse dann zum Client zurückgesendet werden. So ein Vorgehen ist bei Systemen zu beobachten, bei dem die Clients über eine begrenzte Rechenleistung verfügen und deswegen rechenintensive Berechnungen zu dem Server delegieren.

**Dynamischer Methodenaufruf über CORBA** Hier wird der Ansatz des dynamischen Methodenaufrufs diskutiert. Im Gegensatz zu dem statischen Methodenaufruf liefert der dynamische Ansatz eine flexiblere Umgebung. Er erlaubt das Hinzufügen neuer Klassen zu dem bereits bestehenden System, ohne irgendwelche Änderung der Clients durchführen zu müssen. Im Folgenden werden die Vorteile vorgestellt, welche der dynamische Ansatz mit sich bringt. Das sind:

**Leicht erweiterbar und wartbar** Wie eben schon angedeutet müssen bei einer Änderung des Dienstes die Clients nicht angeglichen und neu übersetzt werden. Durch diese Tatsache ist also eine Änderung, Verbesserung, Erneuerung des Dienstes jederzeit möglich. Somit kann man sagen, dass durch diese Entkopplung des Clients von dem Dienst die spätere Dienstwartung und Pflege erheblich erleichtert, wenn nicht sogar gefördert wird.

**Offenheit** Der wichtigste Vorteil hierbei ist die Entwicklung eines Dienstes, bei dem ohne irgendwelche Einschränkungen dieser mit einer nach außen sichtbaren Schnittstelle entwickelt werden kann. Dieses hat zur Folge, dass der Dienst alle benötigten Methoden liefern kann, was bei einer standardisierten Schnittstelle nicht der Fall sein würde, da man einer Einschränkung unterzogen ist.

**Anpassbar** Durch die Offenheit, die der dynamische Methodenaufruf mit sich bringt, ergibt sich noch ein weiterer Vorteil auf der Clientseite. Meldet sich ein Client bei dem Server an, braucht er vorher fast kein Wissen über die darauf befindlichen Dienste zu besitzen. Erst bei einer erfolgreichen Anmeldung kann der Client wahlweise die Informationen über die jeweiligen Dienste holen. Bezieht man diesen Vorteil auf die Mobilität, kann man sagen, dass diese hierdurch erheblich begünstigt wird. Durch die eben genannte Offenheit ist der Client nicht auf eine Ausprägung der Umgebung gebunden. Durch diese jederzeit unterstützte Kontexterkennung kann der Client seine Umgebung jederzeit wechseln und sich an eine neue anpassen.

**Entkoppelt** Die Änderung des Dienstes hat keine Auswirkungen auf den Client, was zur Folge hat, dass die älteren Clientprogramme ohne vorherige Angleichung weiterhin sich

an den neuen Dienst anmelden können. Lediglich ist etwas Aufwand auf der Serverseite notwendig, um die neuen Dienstmethoden sichtbar für den Client zu machen. Ein positiver Aspekt, welcher sich in diesem Zusammenhang ergibt, sind die Clientkosten. Da sich wohl viel mehr Clients an deutlich weniger Dienste anmelden werden, wäre ein Kompatibilitätsverlust deutlich bei den Clientkosten[8] zu spüren.

**Unterstützung der Wiederverwendung** Bereits bestehende Dienste müssen nur an die dynamische Schnittstelle angeglichen werden, wodurch die bestehenden Clients auf diese sofort zugreifen können.

Im Folgenden werden die Nachteile des dynamischen Methodenaufrufs vorgestellt:

**Hohe Performancekosten** Wie das in den meisten anderen Technologien der Fall ist, ist auch bei Corba das dynamische Verhalten mit hohen Kosten an Performance zu verzeichnen. Es werden zusätzliche Corbadienste für das Auffinden der einzelnen Methoden in Anspruch genommen, was das System verlangsamt.

**Kompliziertere Clients** Clientprogramme müssen so weit erweitert werden, dass sie im Stande sind, die richtigen Methoden zu finden und auszuwählen d.h., es kommt zusätzlicher Programmcode hinzu, was den Client wieder komplizierter macht.

**Fehlinterpretation** Was auch eine Fehlerquelle sein kann, ist die falsche Interpretation der Methodensemantik. Die Methodennamen werden von dem Entwickler des Dienstes frei gewählt, was möglicherweise nichts mit der Aufgabe der Methode zu tun hat. Bei einer standardisierten Schnittstelle ist jede Methode genau definiert.

Der dynamische Ansatz zeichnet sich vor allem durch die hohe Flexibilität aus. Diese erlaubt die Anpassung an die Umgebung zur Laufzeit, wodurch die Dienstschnittstelle sich nicht nur auf einen definierten Methodenumfang beschränken muss, sondern den gesamten Leistungsumfang des Dienstes unterstützen kann, ohne jeweilige Eingriffe in dem Client vornehmen zu müssen. Bei dieser Vorgehensweise verliert man einen großen Teil der Performance an der Suche nach den richtigen Methoden, wodurch der Einsatz bei zeitkritischen Problemen von vornherein ausgeschlossen werden kann. Diese Vorgehensweise ist eher für Systeme gedacht, bei dem der Kontext zur Anmeldungszeit unbekannt ist und die zur Verfügung gestellten Dienste nicht in Echtzeit reagieren müssen. Natürlich kann man durch Verwendung leistungsstärkerer Rechner die Antwortzeit an Echtzeit annähern, trotzdem gibt es noch bei dynamischen Verhalten undefinierte Zustände, bei dem das System keine Antwort liefern kann und somit nicht echtzeitfähig ist.

[8] Gemeint sind hier die zeitlichen Kosten.

### Kommunikationsprotokoll

**XML** Hier sollen die Vor- und Nachteile diskutiert werden, welche sich im Zusammenhang mit dem Einsatz von XML bei der Auffindung der Dienstmethoden ergeben. Dazu wird der Ansatz kurz vorgestellt. Mit Hilfe von XML werden alle Methoden des Dienstes beschrieben. Dieses XML-Dokument dient dann als Schnittstellendefinition für die Clients, die den entsprechenden Dienst nutzen wollen. Als mögliches Szenario ist folgendes zu sehen: Bei der Anmeldung bekommt der Client dieses XML-Dokument zugesendet. Daraufhin wird dieser auf der Clientseite geparst. Danach findet die Ausgabe des Methodenumfangs auf der Client-UI statt. Der User hat dadurch mitbekommen, dass er einen Dienst nutzen kann und sieht die Möglichkeiten, die der Dienst zur Verfügung stellt. Nach der Auswahl einer entsprechenden Methode (Anfrage) wird diese mit den entsprechenden Parametern konstruiert und an den Server gesendet. Die Vorteile sind:

**Flexibilität** Durch die Vereinbarung nur einer Sendemethode, mit deren Hilfe man die Anfragen an den Dienst stellt, gewinnt man ein großes Maß an Flexibilität. Die Schnittstelle wird so weit reduziert, dass nur Methoden für das Versenden und Empfangen übrig bleiben.

**Strukturierung der Daten** XML erlaubt eine komfortable Strukturierung der Daten. Dieses hat zufolge, dass die Suche nach den einzelnen Daten erheblich vereinfacht wird.

**Änderbarkeit** Das Kommunikationsprotokoll zwischen dem Client und Server, welches mit XML definiert wird, kann jederzeit an die neu gestellten Anforderungen angepasst werden. Der eigentliche Vorteil, der hierbei entsteht, ist der, dass bei diesen Anpassungsaufgaben Client- und Serverprogramm nicht angefasst werden müssen.

**Hohes Abstraktionsniveau** Mit Hilfe von XML ist man in der Lage ganze Objekte zu beschreiben, die später mit XSL in ein entsprechendes Darstellungsformat transformiert werden können. Wird dieser Weg weiterverfolgt und entsprechend weiter entwickelt, so kann man eine saubere Darstellung eines Objekts im beliebigen Format erreichen und so weit generalisieren, dass diese auf jedem Endgerät darstellbar sind. Hierdurch wird ein hohes Abstraktionsniveau erreicht.

Die Nachteile sind:

**Parser erforderlich** Als Nachteil kann man sicherlich die notwendige Verwendung eines Parsers, der die XML-Datei auswertet, nennen. Da die Rechenleistung auf dem PDA begrenzt ist, nimmt das Parsen des XML-Dokumentes einen großen Teil der Ressourcen in Anspruch, was zum Nachteil für andere, zu gleicher Zeit laufende, CORBA-Dienste und Programme wird. Dadurch wird der Client immer rechenintensiver, benötigt zusätzlich Software (FATCLIENT), was zu Folge hat, dass der nicht überall zum Einsatz kommen kann.

**Möglicher Semantikverlust** Die Rückgabedaten sind so flexibel verpackt, dass deren Semantik verloren gehen kann. In dem XML-Dokument werden Daten mit neuen von jedem Dienst unterschiedlich gewählten Markierungen strukturiert, sodass deren Interpretation sehr schwierig ist. Dadurch ist eine flexible Darstellung aller verfügbaren Dienste sehr schwierig. Somit wird die gewonnene Flexibilität auf der Schnittstellenebene durch die Verlagerung des Interpretationsproblems auf den Benutzer erreicht. Dieser wird angewiesen, die verschiedenen Dienste richtig zu interpretieren und zu benutzen, was eine Fehlerquelle darstellt.

Der oben geschilderte Anwendungsfall zeichnet sich durch eine sehr hohe Flexibilität des gesamten Systems aus. Durch das Voraussetzen nur einer Sendemethode, die alle möglichen Ausprägungen einer Nachricht auf der Basis des XML-Dokumentes sendet, ist man bei der Anwendungsentwicklung vollständig von der Implementierung des Dienstes entkoppelt. Der Client hat als Aufgabe, die Darstellung des Funktionsumfanges des angesprochenen Dienstes zu ermöglichen. Dabei parst der Client das gesendete XML-Dokument und stellt es in einer darstellbaren Form dar. Auf der anderen Seite muss dieser die Benutzereingaben richtig in ein XML-Dokument einpacken und an den verbundenen Dienst senden. Da die Kommunikation zwischen dem Client und dem Dienst so einfach aufgebaut ist und der Client selbst einfach gehalten werden soll, liegt somit die Hauptverantwortung der richtigen Benutzung des Dienstes bei dem Benutzer. Dieser muss den Funktionsumfang des Dienstes richtig deuten und auswählen können, sodass der Dienst die Anfragen richtig bearbeiten und beantworten kann. Aus der Sicht der Softwareergonomie ist dieses aber dem Benutzer nicht zuzumuten. Es würde zu Folge haben, dass, um den Client benutzen zu können, der einzelne Benutzer vorher in deren Benutzung geschult werden müsste. Dieses Szenario entspricht nicht der Anforderung, die am Anfang im Abschnitt 2.4.2 gestellt wurden. Hinzu kommt noch, dass bei der Benutzung der Clients keine Validierung möglich wäre, die prüft, ob der Client die Funktionen in richtiger Reihenfolge aufruft. Dadurch würden viele Korrekturaufrufe von dem Benutzer entstehen, was ebenfalls die Softwarequalität verschlechtert und die Zeit der Benutzung des Dienstes unnötig verlängert. Als letztes Manko dieses Ansatzes ist die Darstellung des Funktionsumfanges. Wie soll von Dienst zu Dienst unterschiedlicher Funktionsumfang richtig dargestellt werden. Dieser muss für den Benutzer verständig und geordnet dargestellt werden, die geschilderte Darstellung ist aber weit entfernt davon. Wenn man noch eine begrenzte Rechenleistung zur Verfügung hat, kann das Parsen des XML-Dokuments zu einer Verschlechterung des Laufverhaltens der andere, parallel laufenden Anwendungen führen. Alles im Ganzen hat dieser Ansatz ohne eine Erweiterung und Beseitigung einiger Schwachstellen zu viele Schwächen, die für eine weitere Verwendung zu gravierend sind.

Eine Erweiterung des Ansatzes könnte wie folgt aussehen. Da die Hauptschwächen bei der Darstellung und Benutzerfreundlichkeit liegen, werden diese als erste betrachtet. Hier kommt die zuvor besprochene XML Stylesheet Technologie zum Einsatz. Mit deren Hilfe wird die Darstellung und Benutzung der Dienste deutlich erleichtert, was auf der anderen Seite die Entwicklungskosten des einzelnen Dienstes aber erhöht. So wird bei der Dienstentwicklung

ebenfalls ein Stylesheet für diesen definiert. Dieser wird mit dem XML-Dokument ebenfalls an den Client gesendet. Der Client wiederum benutzt das Stylesheet, um das gesendete XML-Dokument in einer für den jeweiligen Dienst abhängigen Darstellung und für den Client verständliches Format zu konvertieren, sodass dieser Dienst verständlich für den Benutzer dargestellt werden kann. Als mögliche Erweiterung des Ansatzes kann die Verwendung des *Data Binding* Konzepts in Betracht gezogen werden. Hierdurch wird es möglich Objekte in ein XML-Dokument zu konvertieren und umgekehrt. Damit kann eine flexible Transformation der Daten in definierte Sichten erreicht werden. Für weitere Informationen (s.z.B. SUN, 2004a).

### 2.4.4 Fazit

Viele der Ansätze die hier betrachtet wurden, zeichnen sich durch viele positive als auch negative Merkmale aus, die für den Einsatz in dem hier realisierenden Konzept ausschlaggebend sind. An dieser Stellen sollen diese kurz dargestellt werden und darauf folgend eine Auswahl des hierfür am besten geeigneten getroffen werden. Diese Betrachtung sollte in Hinsicht auf die hier behandelte Aufgaben geschehen.

Web Services stellen eine neue Möglichkeit dar verteilte Systeme zu Realisieren. Durch ihren hauptsächlich XML-basierten Aufbau, ist eine hohe Abstraktion der Kommunikationsseiten gegeben. Hierdurch gewinnt man ein hohes Maß an Interoperabilität und Plattformunabhängigkeit. Leider werden diese Vorteile wieder durch die Unausgereiftheit nivelliert, was diese hierdurch in der weiteren Betrachtung ausschließt, da diese Eigenschaften existenziell für den hier verfolgten Lösungsansatz sind. Es ist abzuwarten wie sich diese weiter entwickeln werden um gegebenenfalls diese einer neuen Analyse zu unterziehen.

Im Weiteren wurden die beiden Plattformen und ihre Merkmale auf die Eignung hin untersucht. Dabei stellten beide interessante Ideen zur Verfügung, die Hilfreich für die Realisierung der mobilen Multimedia Middleware wären. J2EE hat die meisten in dem Abschnitt 2.4.2 gestellten Anforderungen erfüllt und somit sich als eine geeignete Plattform für das hier verfolgte Konzept erwiesen. .NET hat neue Möglichkeiten zur Verfügung gestellt, die J2EE nicht bieten konnte wodurch das Konzept noch weiter erweitert werden konnte. Auf der anderen Seite hat .NET die bei J2EE schon lange enthaltenen und bewerten Möglichkeiten nicht bereitgestellt. Die Plattformabhängigkeit, das Fehlen der Location Transparency, die fehlende Vorgabe eines Komponentenmodels, dass alles was für die Realisierung der hier gestellten Aufgabe notwendig ist, fehlt bei .NET wodurch diese sich hierfür als nicht geeignet herausstellt.

Wie bereits erwähnt stellt CORBA zwei Arten des Methodenaufrufes der Objekte bereit. Der dynamische Aufruf macht die CORBA-Infrastruktur im großen Maße flexibel und verleitet den darauf basierenden Anwendungen Anpassungsdynamik. Die Möglichkeiten, wie die kontinuierliche Evolution eines Dienstes und automatische Anpassung an den Dienstkontext, sind die Merkmale die eine dienstorientierte Architektur im Idealen auszeichnen. Auf der anderen Seite bereitet diese Offenheit vergleichbare Probleme wie das auch bei

der Web Service Technologie der Fall ist. Um diese in den Griff zu bekommen, mussten auf der Client- und Serverseite Mechanismen hinzukommen, wodurch diese unnötig aufgebläht werden würden, was zumindest bei dem Client nicht gewünscht ist. So stellt CORBA auch den statischen Ansatz des entfernen Methodenaufrufes bereit der hier Abhilfe schaffen konnte. Diese hat den Vorteil, dass durch diesen eine vorgegebene standardisierte Schnittstelle gegeben sein wird, auf der anderen Seite auch noch sehr effizient ist. Das Problem der Erkennung des Dienstkontextes wird an das Kommunikationsprotokoll verlagert. Somit wird in dieser Betrachtung der CORBA-Ansatz mit dem von diesem zur Verfügung gestellten statischen Methodenaufruf als der für die Realisierung der hier gestellten Aufgabe best geeigneter ausgewählt.

Wie bereits erwähnt, wird ein XML basiertes Kommunikationsprotokoll verwendet um unter anderem die Dienstkontextinformationen dort ablegen zu dürfen. XML bringt aber noch weitere Vorteile mit sich. Durch die Verwendung von XML werden wie in dem WS die beiden Kommunikationsseiten voneinander entkoppelt. Hierdurch werden diese offener gegenüber Änderungen bzw. Erweiterungen. Im Weiteren wird die Anpassungsfähigkeit des Kommunikationsprotokolls an spezielle Probleme deutlich erleichtert. Alles im Ganzen ergänzt der Einsatz von XML die Stärken des statischen Methodenaufrufs von CORBA, sodass diese zusammen die notwendigen Vorraussetzungen für den Lösungsansatz erfüllen.

# Kapitel 3

# Entwurf

„Die Grenzen der eigenen Fähigkeiten erweitern bedeutet, neue Dinge tun zu können."

Gerda Alexander

## 3.1 Einleitung

An dieser Stelle wird die Entwurfsphase eingeleitet. Ausgehend von dem Kapitel Analyse und deren Erkenntnissen, sollen die Softwareanforderungen der hier zugrunde liegenden Softwarekomponenten in einen softwaretechnischen Entwurf umsetzen werden. Aus dem hier gefertigten Entwurf soll eine saubere Grundlage für die nachfolgende Implementierungsphase geschaffen werden.

Die hier anstehenden Aufgaben konzentrieren sich auf den Entwurf der mobile Multimedia Middleware und weniger auf die darauf aufbauenden Multimedia Dienste. Die Middleware bildet das Herzstück der oben besprochenen Architektur. Hierdurch entstehenden exakten Vorgaben für die Implementierungsphase schließen die Ausarbeitung der Software- sowie Hardwarearchitektur ein. Außerdem beinhaltet diese die Gestaltung der einzelnen Softwarekomponenten. Als Ergebnis ist am Ende ein näher analysierter und spezifizierter Aufbau der Software zu erwarten.

In diesem Kapitel wird als Erstes die Umgebung sowie deren Randbedingungen ermittelt und festgelegt. Als Nächstes werden konzeptionelle Entscheidungen in Hinsicht auf die benötigten Hilfssysteme getroffen. Im nächsten Abschnitt wird zu dem tatsächlichen Entwurf der Softwarearchitektur übergegangen. Für die nun definierte Softwarearchitektur wird im darauf folgenden Abschnitt der Funktions- und Leistungsumfang der Systemkomponenten festgelegt, sowie deren Schnittstellen.

## 3.2 Umgebungs- und Randbedingungen

An dieser Stelle sollen die Randbedingungen genannt werden, die in Rahmen dieser Diplomarbeit und der erfolgreichen Fertigstellung der Software zu beachten sind. Diese ergeben

sich aus den in früherem Kapitel gestellten Anforderungen und werden hier nach Software und Hardware unterteilt.

### 3.2.1 Software

In erster Linie wird auf jedem Rechner, auf dem die mobile Multimedia Middleware laufen soll, eine *Java Virtuelle Maschine*, kurz JVM benötigt. Diese muss mindestens der Java-Version 1.1.8 entsprechen.

Für die auf einer höheren Ebene angesiedelte Kommunikation wird ebenfalls eine CORBA Implementierung benötigt. Durch diese Kommunikationsart wird die Erstellung der dynamisch verteilten Anwendung vereinfacht, da tiefer gehendes Wissen über Netzwerkkommunikation nicht erforderlich ist. Hinzu kommt noch, dass häufig gebrauchte Dienste bereits in einer erprobten Implementierung vorliegen, was den Aufwand und die Fehleranfälligkeit mindert.

Das hier erstellte System soll in der Anfangsphase ein Ein-Benutzersystem unterstützen. Diese Randbedingung gilt nur für die Prototypen, die hier entwickelt werden sollen. In der Realität ist das System nur dann sinnvoll, wenn mehrere Benutzer gleichzeitig auf verteilte Ressourcen zugreifen können. Dabei muss dann die Problematik der Zugriffskontrolle genauer betrachtet werden, was in Rahmen dieser Arbeit nicht möglich ist.

### 3.2.2 Hardware

Für die erfolgreiche Implementierung der Software wird ein mobiles Gerät benötigt, welches über einen Funkmodul eine Verbindung mit anderen Rechnern aufbauen kann, wobei dieses das TCP/IP Verbindungsprotokoll unterstützen soll. Grundsätzlich wird keine Anforderung an die Funkkommunikation gestellt, sodass das Funkmodul ausgetauscht werden kann. Somit können mehrere Funktechnologien unterstützt werden wie z.B. Bluetooth oder W-Lan. Da aber in der Definition der Anwendungsfälle die Rede von ortsrelevanten[1] Diensten ist, wird hier die Bluetooth Funktechnologie bevorzugt eingesetzt.

## 3.3 Konzeptionelle Entscheidungen

An dieser Stelle sollen alle benötigten Hilfssysteme identifiziert werden, die für die Realisierung der Software notwendig sind. Dabei wird untersucht, ob diese bereits vorhanden sind oder selbst entwickelt werden müssen.

[1] Dienste die innerhalb einer geringen Entfernung zu benutzen sind und somit an den Ort, bzw. ortsrelevanten Dienstleistung gebunden sind.

### 3.3.1 Datenhaltung

#### Middleware

In den meisten Fällen benötigt eine Anwendung eine Datenhaltung. So werden wichtige Daten langfristig persistent gemacht und können jederzeit wieder aufgerufen werden. Aus der Sicht der mobilen Multimedia Middleware ist die Datenhaltung als solches keine Funktionalität, die von dieser bereitgestellt werden soll. Vielmehr steht dieses in dem Verantwortungsbereich der Entwickler der Multimedia Dienste, für die entsprechende Datenhaltung zu sorgen. Die Begründung liegt in dem breiten Einsatzgebiet. Es können viele verschieden Multimedia Dienste entwickelt werden. Diese könnten jeweils unterschiedliche Anforderungen an die Datenhaltungsschicht haben. Aus diesem Grund wird die mobile Multimedia Middleware an keine Persistenzschicht gebunden um somit offen für jede Dienstimplementierung sein.

Bei der mobilen Multimedia Middleware entstehen nur zwei Arten von Daten, die persistent gemacht werden müssen. Zum einen sind das die im Betrieb entstehenden Protokolldaten, die für die Überwachung der mobilen Multimedia Middleware notwendig sind und deren Zustand kontinuierlich protokolliert werden muss. Zum anderen müssen einige Objektzustände persistent gemacht werden, damit diese bei Bedarf wiederhergestellt werden können.

Die Datenhaltung, die nur für die mobile Multimedia Middleware relevant ist, wird in diesem Fall aus Einfachheitsgründen mit Hilfe von simplen Dateien realisiert. Im Genaueren wird dazu die Dateiverwaltungsfunktionalität des Betriebssystems in Anspruch genommen. Dieses Konzept deckt alle erforderlichen Datenhaltungsansprüche ab, die die mobile Multimedia Middleware stellt.

#### Client-Server

Um die Funktionalität der mobilen Multimedia Middleware praktisch verdeutlichen zu können, werden ebenfalls Prototypen als Demonstratoren für einen Multimedia Dienst und Client entwickelt. Diese sind einfacher Natur, sodass auch die Aspekte der Datenhaltung in den Demonstratoren einfach ausfallen werden. Begründung hierfür ist, dass die Entwicklung der Prototypen nicht Gegenstand dieser Arbeit ist. Dennoch wird darauf hingewiesen, dass die beiden Kommunikationsseiten bzw. dort notwendige Datenhaltung beliebig kompliziert werden kann. In der mobilen Architektur werden diese stark modularisiert, sodass deren Datenhaltung nach außen Transparent ist und somit keine Auswirkungen auf die mobile Multimedia Middleware hat.

### 3.3.2 Verteilung im Netz

An dieser Stelle soll die Verteilung der mobilen Multimedia Middleware im Netzwerk festgelegt werden. Im Wesentlichen gibt es zwei Verteilungsarten, die Client/Server und die Web Architektur. Die Web Architektur kommt hier nicht in Frage, da dort die gesamte Anwendung

auf der Serverseite vorgesehen ist. Dieses entspricht aber nicht den Anforderungen, da eine Umverteilung von Teilen der mobilen Multimedia Middleware auf dem Client ermöglicht werden soll. Hinzu kommt, dass der Einsatz eines Web Browsers in diesem Fall die Entwicklung der Dienste im gewissen Maß einschränkt. Ein weiteres Merkmal dieser Architektur ist die Verbindungsart, welche in der Regel nur temporär ist. Dieses wird aber als Hindernis für einige Dienstimplementierungen angesehen, im Genaueren bei den Implementierungen die Echtzeitfunktionalität anbieten wollen.

In Frage kommt also die erste Architektur, die Client/Server Architektur. Diese erlaubt die implizite Verteilung der mobilen Multimedia Middleware auf die beiden Kommunikationsseiten. Hinzu kommt die andere Art der Kommunikation. Zwischen den Client und Server besteht grundsätzlich eine permanente Verbindung. Hierdurch ist eine Benutzersitzung einfacher zu verfolgen.

Nachdem die Verteilungsarchitektur im groben fest steht, sollte deren interne Verteilung weiter diskutiert werden. Bei dem oben gewählten zweistufigen Client/Server-Konzept kann die Verteilung der Darstellungsschicht, Anwendungsschicht und Persistenzsicht in fünf verschiedenen Verteilungsvarianten unterschieden werden. In diesem Zusammenhang treten solche Schlagwörter wie: Fat- bzw. Thin-Client auf. Dabei werden die einzelnen internen Komponenten jeweils zwischen Client und Server verschoben. So wird ein Client Fat-Client genannt, wenn dieser mehr Programmlogik implementiert als der Server. Und umgekehrt, wenn der Client den kleineren Teil der Anwendung implementiert, so wird dieser Thin-Client genannt. Im Fall der mobilen Multimedia Middleware ist diese interne Verteilung, bzw. deren Variation nicht möglich, da sich in diesem Fall um eine symmetrische Schichtenarchitektur handelt, bei der ein Schichtenmodell mit linearer Ordnung[2] implementiert wird. Im Fall der Multimedia Dienste ist die Variierung der Verteilung möglich.

In der Abbildung 3.1 wird die grobe Schichtenarchitektur der Anwendung dargestellt. Daraus ist die resultierende logische, sowie physische Schichtenverteilung der Anwendung zu erkennen. Es werden drei logische Schichten, die Kommunikationsschicht, Anwendungsschicht und die Dienstimplementierung identifiziert. Dabei ist zu beachten, dass die Kommunikationsschicht im großen Maße auf der CORBA Middleware basiert und somit diese als ein Teil der Kommunikationsschicht anzusehen ist. Diese logischen Schichten stellen die Modularisierungseinheiten der Anwendung dar.

Wie das aus der Abbildung hervorgeht, stellen die Anwendungs- und Kommunikationsschicht die eigentliche mobile Multimedia Middleware dar. Diese stellt eine Infrastruktur für die jeweiligen Dienstimplementierungen bereit. Eine weitere Betonung auf eine Schichtenarchitektur stellen die in der Zeichnung dargestellten logischen Kommunikationswege zwischen den gleichnamigen Schichten zwischen dem Client und dem Server dar. So wie bei ISO/OSI Schichtenmodel kommunizieren diese jeweils mit einander auf der jeweiligen Ab-

[2] Ein Schichtenmodell mit linearer Ordnung liegt vor, wenn von einer Schicht immer nur auf die nächst niedrigere Schicht zugegriffen werden kann. Beispiele für Schichtenmodelle mit linearer Ordnung sind das ISO-OSI Schichtenmodell, TCP/IP (Balzert, 2000).

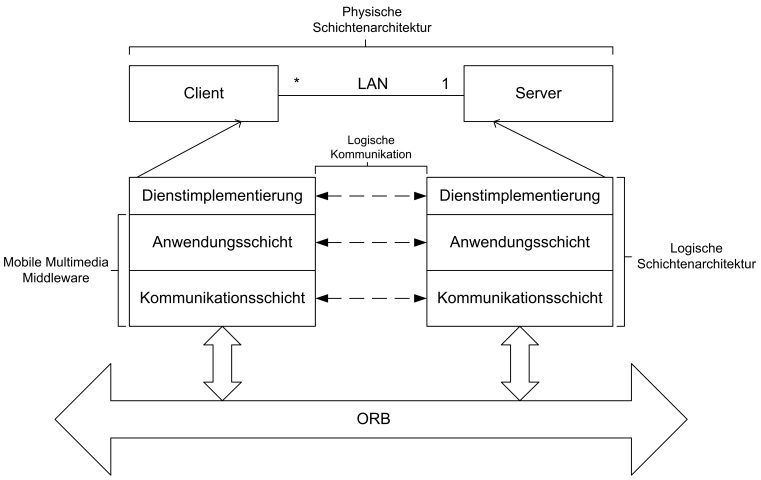


Abbildung 3.1: Grobe Schichtenarchitektur

straktionsebene.

Ebenfalls kann man die physische Schichtenarchitektur aus der Abbildung 3.1 entnehmen. Wie zuvor bereits erwähnt, wurde hier das Client/Server Verteilungsmuster vorgezogen. Die hieraus resultierende physische Schichtenarchitektur sieht vor, dass auf den beiden Kommunikationsseiten die gleichen Anwendungskomponenten notwendig sind.

### 3.3.3 Benutzungsoberfläche

#### Mobile Multimedia Middleware

Die Betrachtung der Problematik der Benutzeroberfläche der mobilen Multimedia Middleware wird sehr kurz ausfallen, da diese nicht als Anforderung im Vordergrund steht. Die Aufgabe dieser Middleware ist eine Umgebung für Multimedia Dienste zu schaffen und weniger Vorgaben in der Darstellung deren zu stellen. Durch Auslagerung dieser Problematik an die Dienste selbst bleibt die Kommunikationsmiddleware weiterhin weitgehend offen, sodass hierdurch keine Einschränkungen an die Dienstimplementierung in der Hinsicht gestellt werden.

Die Benutzersicht der mobilen Multimedia Middleware selbst wird textbasiert im Kommandofenster realisiert. Dabei wird versucht hierdurch nur die Administrationsfunktionalität, sowie ein eingeschränktes Loggen zu ermöglichen. Damit wird eine einfache Benutzungsschnittstelle zur der Kommunikationsmiddleware entstehen.

#### Client-Server

Für die beiden Client und Server Prototypen wird ebenfalls eine Benutzungsoberfläche entwickelt. Die entstehenden Oberflächen werden sehr einfach ausfallen. Diese sollten nur die In-

teraktionen zwischen den beiden Kommunikationspartnern anzeigen und weniger eine nach dem softwareergonomischen[3] Aspekten entwickelt werden, da die Umsetzung dieser Anforderung den Rahmen dieser Arbeit sprengen würden. Zu betonen ist aber, dass diese beliebig kompliziert entwickelt werden können und trotzdem weiterhin die uneingeschränkte Kommunikationsunterstützung der mobilen Multimedia Middleware genießen würden.

### 3.3.4 Middleware

Ein guter Ansatz ist es, möglichst viele Dienstleistungen auf hohem Abstraktionsniveau von anderen Systemen in Anspruch zu nehmen. So soll es auch in diesem Fall sein. Der hier verfolgte Lösungsansatz verwendet eine Implementierung des von der OMG[4] definierten OMA[5] Architektur.

Durch den Einsatz dieser Middleware wird eine sprachunabhängige Plattform für Multimedia Dienste geschaffen. Zusätzlich wird durch CORBA[6] die interne Kommunikation auf einer höheren Abstraktionsebene durchgeführt. Des Weiteren bring CORBA einige standardisierte und bereits bewährte Hilfsdienste mit, die in diesem Lösungsansatz verwendet werden.

### 3.3.5 XML

XML wird hier in der Kommunikationsschicht zum Einsatz kommen. Durch diese wird eine Strukturierung der zu versendenden Kommunikationsdaten gegeben. Die Daten werden in wohl definierten XML-Dokumenten abgespeichert, was einen zusätzlichen Kontrollmechanismus darstellt. Außerdem wird durch die Versendung der XML-Dokumente die Interoperabilität begünstigt[7]. Dafür ist an den beiden Kommunikationsseiten jeweils ein Parser erforderlich, um die Daten aus dem XML-Dokument entnehmen zu können.

XML hat in der Middleware die Aufgabe des Strukturmappings. D.h. diese abstrahiert die Methodenaufrufe, Objektaufrufe und bietet die Möglichkeit, Strukturen (wie Dienstleistungen), die sich selbst beschreiben, zu versenden.

## 3.4 Entwurf der Software-Architektur

An dieser Stelle wird die Struktur des Softwaresystems der mobile Multimedia Middleware durch Systemkomponenten und ihre Beziehungen untereinander beschrieben. Die daraus

[3] Nach Balzert (2000) ist Software-Ergonomie die Regeln zur Gestaltung eines Software-Arbeitsplatzes, d.h. der Anwendungssoftware und des Arbeitsplatzes. Das Ziel dieser ist, dem Benutzer in seinem Nutzungskontext ein gebrauchstaugliches Software-Produkt zur Verfügung zu stellen. Gebrauchstauglichkeit gliedert sich in die Kriterien: Effektivität, Effizienz und Zufriedenstellung.

[4] Object Management Group; (sehe http://www.omg.org)

[5] Object Management Architecture (OMG, 2004b)

[6] Common Object Request Broker (OMG, 2004a)

[7] Wie das z.B. bei Web Services zu beobachten ist.

entstehende Softwarearchitektur soll dann als Referenz bei der Implementierung dienen.

Im Folgenden werden die aus der Analyse kommenden Anwendungsfälle noch weiter verfeinert. Das daraus entstehende System wird in Systemkomponenten zerlegt. Danach wird eine Strukturierung der Systemkomponenten des Systems in ein geeignetes Systemmodell vorgenommen. Nach der Festlegung der Anordnung der Systemkomponenten werden deren Beziehungen zu einander beschrieben.

### 3.4.1 Überarbeitung der Anwendungsfälle

An dieser Stelle sollen die Anwendungsfallbeschreibungen komplettiert und überarbeitet werden.

#### Verbindung mit einen Dienstserver

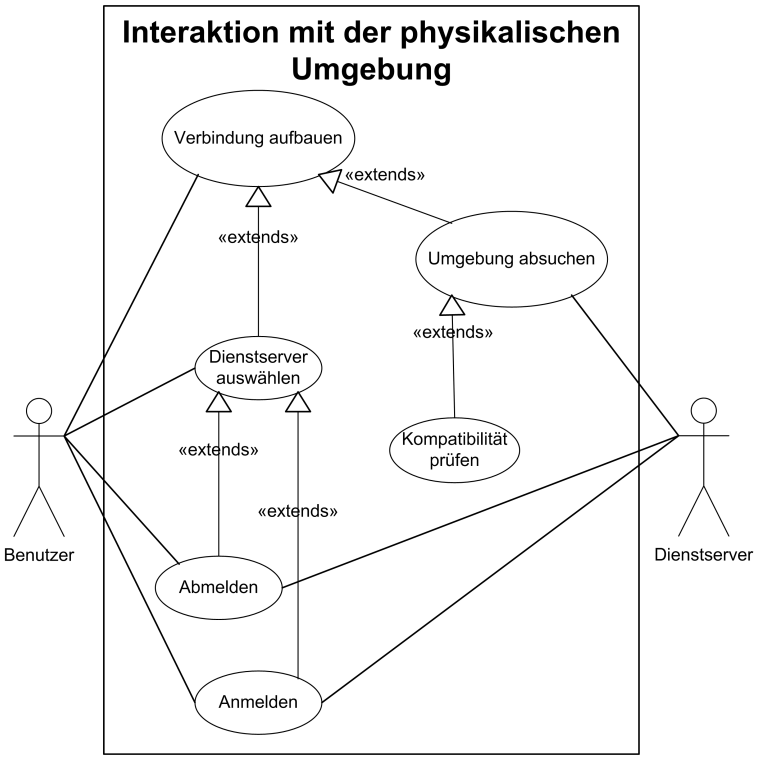


Abbildung 3.2: Anwendungsfalldiagramm: Benutzerinteraktionen mit der physikalischen Umgebung.

Ein PDA Nutzer, welcher auf Dienste in der Umgebung zugreifen will, muss sich als Erstes in der Umgebung kenntlich machen. D.h., auf Aufforderung soll der PDA die Funkschnittstelle aktivieren und die Umgebung nach geeigneten Funknetzzugangspunkten absuchen. Wurde

einer gefunden, sollte als nächstes die Prüfung der Kompatibilität mit der auf dem PDA residierenden Zugangssoftware veranlasst werden. Ist die Prüfung erfolgreich verlaufen, wird der Server für den PDA Benutzer als Auswahl angeboten. Nachdem die Umgebung einige Zeit abgesucht wurde und mindestens ein Dienstserver in der Umgebung gefunden wurde, wird dem Benutzer die Auswahlmöglichkeit zwischen den verschieden Servern gegeben. Der Benutzer wählt einen aus und meldet sich dort an. Ab jetzt ist er im Stande alle verfügbaren Dienste abzurufen. Nachdem die Nutzung des Servers nicht mehr relevant ist, kann der Benutzer sich bei diesem abmelden, wodurch auf beiden Seiten Ressourcen freigegeben werden können. Der eben geschilderte Anwendungsfall wird noch einmal grafisch in dem Anwendungsfalldiagramm 3.2 dargestellt.

Aus Sicht der Mobilität ist dieser Anwendungsfall aber weiter zu bemängeln. Dieser hat einen Schwachpunkt, durch den die Mobilität im eigentlichen Sinne nicht gewährleistet wird. Der Benutzer ist darauf angewiesen, jeweils den Dienstserver, auf dem die Dienste residieren, auszuwählen und sich bei ihm anzumelden. Dieses ist aber möglicherweise für den erstens uninteressant und zweitens erhöht sich hierdurch die Komplexität der Benutzerführung, was die Benutzerfreundlichkeit mindert.

In diesem Fall tritt der PDA selbst als ein Akteur auf und stellt die auf der höheren Abstraktionsebene positionierte Dienstumgebung bereit. Gemeint ist an dieser Stelle, dass der Benutzer sich nicht mehr um die Problematik der physikalischen Umgebung kümmern muss, sodass diese für ihn transparent ist, und dieser ein Eindruck bekommt, als würde die Umgebung nur aus Multimedia Diensten bestehen.

Es existiert aber eine Benutzergruppe, bei dem dieser Anwendungsfall von Bedeutung ist. Gemeint ist ein administrativer Benutzter, welcher bei der Administration genaue Kontrolle über die physikalische Umgebung besitzen muss. Dieser will sich bei einem bestimmten Dienstserver anmelden, bei dem er die Administrationstätigkeit durchführen will.

#### Verbindung mit einen Dienst

Um eine Verbindung mit einem Dienst aufbauen zu können, wird die Verbindung mit einem Dienstserver, auf dem mögliche Multimedia Dienste residieren vorausgesetzt. Diese wurde bereits im Abschnitt 3.4.1 beschrieben.

Ein automatischer Prozess erkennt und stellt die verfügbaren Dienste auf dem PDA bereit. Der Benutzer bekommt also auf dem Bildschirm alle verfügbaren Dienste zu sehen, auf die bzw. auf deren Dienstleistung sofort zugegriffen werden kann. Hierzu ist die Auswahl eines Dienstes und eine Anmeldung bei diesem notwendig. Diese kann einmal von einem PDA-Benutzer manuell oder auch automatisiert ,im Hintergrund, vollzogen werden. Danach wird der Dienst sowie die Clientsoftware initialisiert und für die Nutzung des Dienstes vorbereitet. Der bis jetzt beschriebene Anwendungsfall zeigt das Standardverhalten, welches bei einer Dienstnutzung durch einen Benutzer zu erwarten ist. Dieses kann aber noch weiter erweitert werden.

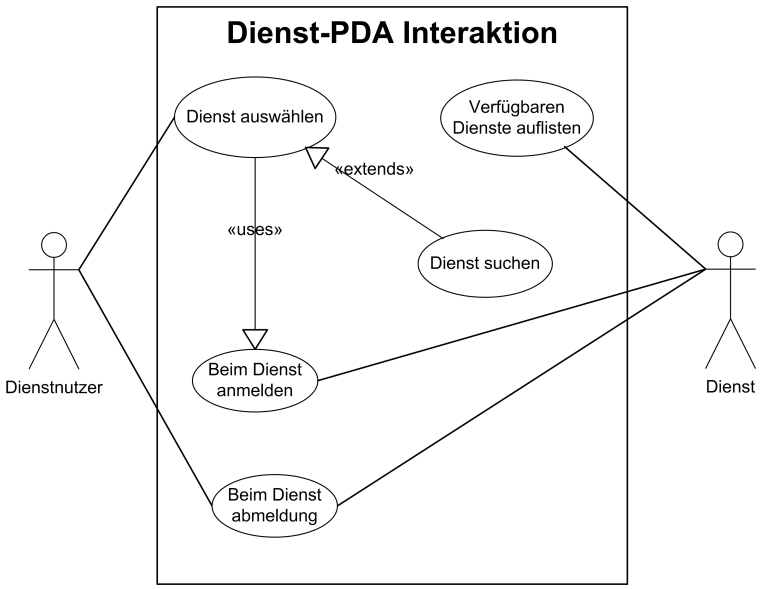


Abbildung 3.3: Anwendungsfalldiagramm: Interaktion zwischen einen Dienst und einen PDA.

So kann bei dem Auswahlprozess durch eine zu hohe Anzahl der verfügbaren Dienste die Auswahl erschwert werden, da die Übersicht verloren gehen kann. Hierzu wird ein zusätzlicher Geschäftsprozess Dienstsuche hinzugefügt, mit dessen Hilfe nach dem geeigneten Dienst gesucht werden kann. Ist der Benutzer mit der Nutzung des Dienstes fertig, kann sich dieser bei diesem abmelden. Das Abmelden ist hier ebenfalls wichtig, weil hierdurch die auf dem PDA knappen Ressourcen wieder freigegeben werden können. Der eben beschriebene Anwendungsfall wird noch einmal in dem Anwendungfalldiagramm 3.3 dargestellt.

**Kommunikationsbehandlung**

Im Kapitel 2.1 wurden einige Probleme genannt, die durch die Verwendung der Funknetztechnologie sowie durch die Mobilität entstehen können. Gemeint ist der Verlust der physikalischen Verbindung zu einen Zugangspunkt sowie der Wiederaufbau dieser. Dem Benutzer kann man die Behandlung dieser Problematik nicht zumuten, sodass diese an die mobile Multimedia Middleware weiter delegiert wird.

In dem Anwendungsfall 3.4 treten als Akteure einmal der Benutzer und die mobile Multimedia Middleware in den Vordergrund. Der Benutzer ist an der eigentlichen Kommunikation interessiert. Bei dieser übernimmt die mobile Multimedia Middleware im Hintergrund die Behandlung der dabei entstehenden Probleme. Hierzu wird die Middleware die Verbindung über deren gesamte Dauer überwachen. Geht die Verbindung ungewollt verloren, so werden die notwendigen Schritte automatisch eingeleitet, um diese wieder aufzubauen. Dabei muss zuerst die physikalische Verbindung zu einem Zugangsknoten hergestellt werden. Ist diese hergestellt, wird bei Notwendigkeit die logische Verbindung gegebenenfalls hergestellt.

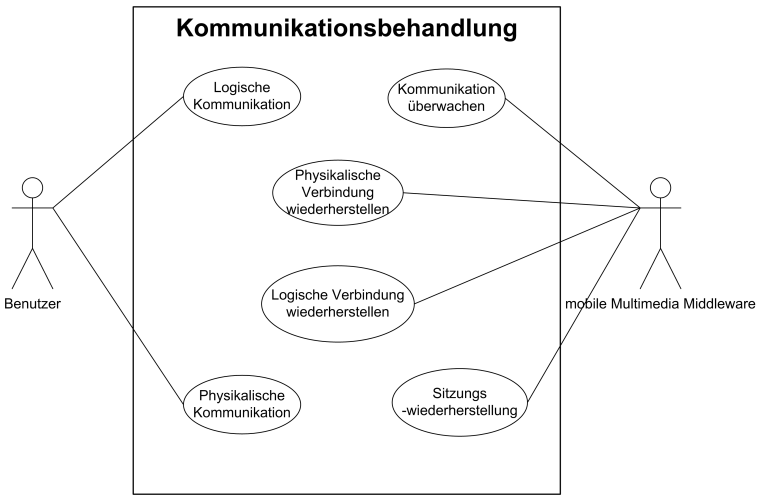


Abbildung 3.4: Anwendungsfalldiagramm: Kommunikationsbehandlung.

Sind beide Verbindung hergestellt, so versucht die mobile Multimedia Middleware zusätzlich, den vorherigen Sitzungszustand wiederherzustellen und damit dem Benutzer die Umgebung bereitzustellen, in der dieser sich vor dem Verbindungsabbruch bewegt hat.

#### Dienstadministration

Eine bereits oben genannte spezialisierte Benutzerrolle ist der Administrator. Dieser hat als Aufgabe, die mobile Multimedia Middleware sowie die darin zur Verfügung gestellten Dienste zu administrieren. Dabei fallen verschieden Tätigkeiten an. So muss in der Anfangsphase ein Dienst bei der mobilen Multimedia Middleware registriert und gestartet werden, damit dieser für Benutzer bereit stehen kann. Es kann aber auch der entgegengesetzte Fall eintreten. Ein veralteter Dienst sollte aus dieser Infrastruktur entfernt werden. Hierbei sollte der entsprechende Dienst sauber beendet werden, was auch die saubere Schließung der noch vorhandenen Sitzungen erfordert. Ist dieser erfolgreich gewesen, kann der Dienst aus der mobilen Multimedia Middleware Umgebung deregistriert werden. Zusammengefasst werden diese Anwendungsfälle in dem Anwendungsfalldiagramm 3.5 dargestellt.

### 3.4.2 Komponenten

An dieser Stelle sollen einige, bis jetzt identifizierte, Komponenten grob beschrieben werden. Diese werden dann in dem Abschnitt 3.4.3 bei der Verfeinerung der Anwendungsfälle zur Hilfe genommen.

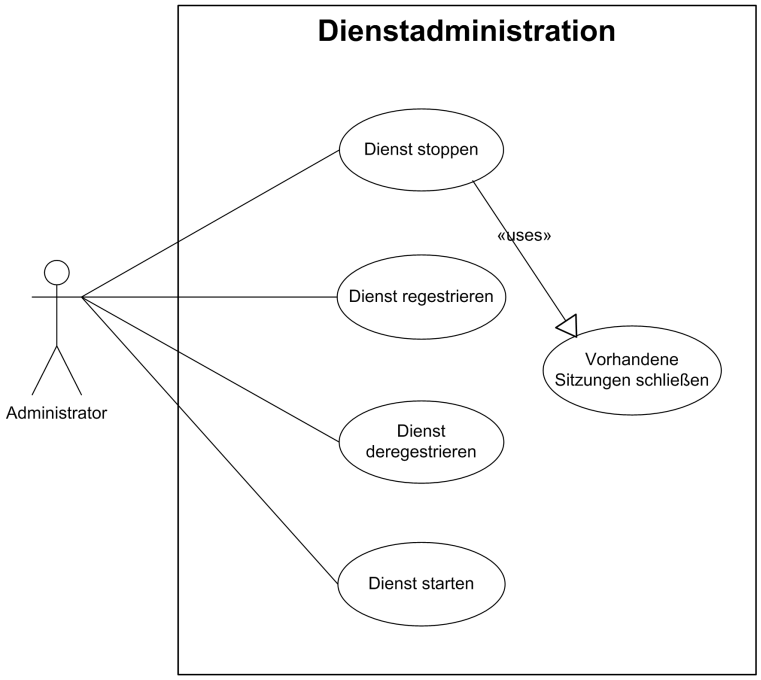


Abbildung 3.5: Anwendungsfalldiagramm: Dienstadminstration.

### Connection

Diese Klasse stellt alle Methoden bereit, die für den Verbindungsaufbau notwendig sind. Zusätzlich hierzu kapselt diese Suchmethoden, durch die die physikalische Umgebung wahrgenommen[8] werden kann. Gleichzeitig ist dieser der Beobachter der beiden Klassen *Server* und *Services*. Bei den beiden genannten Klassen entstehende Kommunikationsprobleme werden sofort von dieser wahrgenommen, sodass diese Maßnahmen treffen kann, um die Verbindung weiter aufrecht zu erhalten.

### Server

Diese Klasse stellt eine Adapter-Klasse für die physikalischen Server dar. Diese stellt einige Methoden zur Verfügung, die notwendig sind, um eine physikalische Verbindung mit einem Server aufbauen zu können. Außerdem werden alle verfügbaren Informationen über den Zugangsserver in dem Serverobjekt gespeichert. Im Falle auftretender Kommunikationsprobleme hilft diese Klasse ebenfalls, mit einigen Hilfsmethoden, dieser aufrecht zu erhalten.

[8] An dieser Stelle sind Zugangsserver mit den darauf residierenden Multimedia Diensten gemeint.

**Service**

Service ist eine weitere Adapter-Klasse, die für einen verbundenen Dienst steht. Der Verbindungsaufbau, sowie die eigentliche logische Kommunikation mit einem Dienst werden in dieser Klasse gekapselt. Die dienstrelevanten Eigenschaften sind auch hier zu finden. Zusätzlich hierzu stellt diese einige Hilfsmethoden bereit, die bei der Wiederherstellung einer ungewollt unterbrochenen Verbindung notwendig sind.

**WirelessAdapter**

Die *WirelessAdapter* Klasse bietet Methoden für die Herstellung einer Kommunikation mit den Funkmodulen, hier auch als Zugangspunkte bezeichnet. Durch diese wird die für die Kommunikation verwendete Funktechnologie von der internen Implementierung der mobilen Multimedia Middleware entkoppelt, sodass ein Austausch dieser jederzeit, ohne Einflussnahme auf die mobile Multimedia Middleware möglich ist.

**ServerListDialog**

Die *ServerListDialog* Klasse bietet ein Dialog für die Darstellung von in der näheren Umgebung gefundenen Zugangsservern an. Dabei werden diese als eine Liste dargestellt, wobei diese nach verschiedenen Sortierverfahren sortiert werden kann.

**ServiceListDialog**

Die *ServiceListDialog* Klasse bietet ebenfalls ein Dialog an. In diesem werden die, in der näheren Umgebung gefundenen Multimedia Dienste, dargestellt. In diesem können verschiedene Filtermethoden auf die Dienstliste angewendet werden, sodass der geeignete Dienst gefunden werden kann.

**ScanThread**

Diese Klasse kapselt die Funktionalität, welche eine automatisierte Erkennung der Umgebung erlaubt. Dabei werden alle verfügbaren Multimedia Dienste aufgespürt und die Informationen über die dort gespeichert.

**ServiceDialog**

Diese Klasse ist eine Stellvertreterklasse, welche dafür sorgen soll, dass die für den ausgewählten Multimedia Dienst benötigte Darstellungsmöglichkeit richtig initialisiert wird.

### 3.4.3 Abbildung der Anwendungsfälle

An dieser Stelle sollten die oben spezifizierten Anwendungsfälle weiter durch Sequenzdiagramme beschrieben werden.

#### Verbindung mit einen Dienstserver

**Manueller Verbindungsaufbau** Als erstes wird der Vorgang der manuellen Anmeldung, bzw. Erkennung der physikalischen Umgebung, mit Hilfe eines Sequenzdiagramms beschrieben.

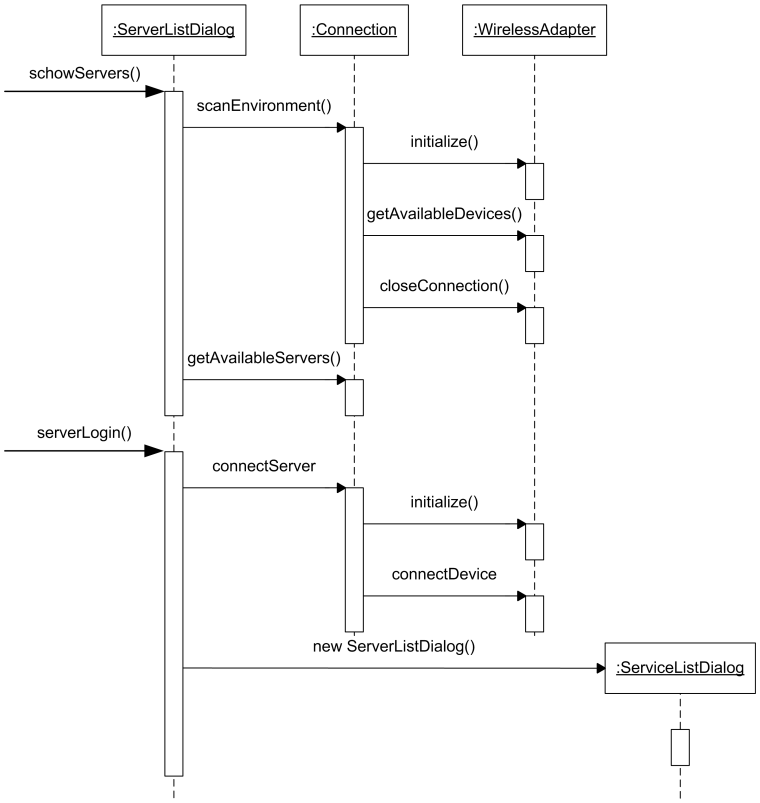


Abbildung 3.6: Sequenzdiagramm: Umgebungsscann und Serverlogin.

Der Vorgang der Erkennung der physikalischen Umgebung wird mit der Methode *showServers()* der Klasse *ServerListDialog* initiiert. Diese ruft die Methode *scanEnvironment()* der Klasse *Connection* auf. Diese Klasse kapselt die Funktionalität, die erforderlich ist, um die nähere Umgebung nach geeigneten Zugangsservern abzusuchen. Im Einzelnen ruft diese die Methoden *initialize()*, *getAvailableDevices()* und *closeConnection()* der Klasse *WirelessAdapter* auf. Mit der Methode *initialize()* wird das Bluetooth Funkmodul initialisiert. Danach wird mit der Methode *getAvailableDevice()* nach geeigneten Bluetooth Zugangspunkten gesucht. Zuletzt wird mit der Methode *closeConnection()* die Verbindung zu den gefundenen

Zugangspunkten gekappt, dabei aber deren Eigenschaften in einer Liste gespeichert. Durch die Verbindungsunterbrechung wird der Energieverbrauch gesenkt, was bei den mobilen Geräten sehr sinnvoll ist.

Nachdem die Umgebung erfolgreich abgesucht wurde, holt sich der *ServerListDialog* die Liste der gefundenen Zugangspunkte und stellt diese in dem Dialog dar. Dieses wird mit der Methode *getAvailableServers()* der Klasse *Connection* bewältigt. Dabei ist zu beachten, dass diese Methode alle Server zurückgibt, die die entsprechende mobile Multimedia Middleware unterstützen. Alle anderen gefundenen Bluetooth Geräte werden in dem Dialog nicht dargestellt.

Nachdem die gefundenen Server in dem Serverdialog für den Benutzer sichtbar gemacht wurden, kann dieser einen Server aus der Liste auswählen und sich mit diesem verbinden. Die Verbindung wird mit der Methode *serverLogin()* der Klasse *ServerListDialog* initiiert. Diese ruft die Methode *connectServer()* der Klasse *Connection* auf, die die dafür erforderliche Funktionalität kapselt. Damit ist die Initialisierung des Funkmoduls und der Verbindungsaufbau mit dem ausgewählten Server gemeint. Dieses wird mit den beiden Methoden *initialize()* und *connectDevice()* der Klasse *WirelessAdapter* bewältigt. Wurde eine Verbindung erfolgreich aufgebaut, so erstellt die Methode *serverLogin()* eine neue Instanz der Klasse *ServiceListDialog*. Der eben geschilderte Vorgang wird in dem Sequenzdiagramm 3.6 grafisch dargestellt.

**Automatisierter Verbindungsaufbau** Wie zuvor in dem Abschnitt 3.4.1 geschildert, ist der Anwendungsfall *manuelle Verbindungsaufbau* nur für eine deutlich kleinere Benutzergruppe sinnvoll. In den meisten Fällen wird aber eine automatisierte Erkennung der Umgebung gefordert. Hierdurch soll die Ebene der Dienstserver für den Benutzer transparent gemacht werden, sodass dieser den Eindruck bekommt, als würde er sich nur in einer dienstorientierten Umgebung bewegen. Hierzu ist eine weitere Klasse erforderlich. Gemeint ist die Klasse *ScanThread*. Wie schon der Name selbst sagt, ist diese Klasse ein Thread, dessen Aufgabe darin besteht, den Dienstsuchvorgang in regelmäßigen Abständen anzustoßen und das Ergebnis mit Hilfe des ServiceListDialogs für den Benutzer sichtbar zu machen. Dieser Vorgang wird mit der Methode *startSearchServices()* der eben genannte Klasse initiiert. Diese Methode überprüft am Anfang, ob bereits eine Instanz der Klasse *ServiceListDialog* existiert. Bei dem ersten Aufruf wird eine Instanz diese Klasse erzeugt, bei weiteren Aufrufen wird die alte Instanz weiter verwendet. Dieses ist erforderlich, damit die Ergebnisse der Suchvorgänge auf dem gleichen Dialog erscheinen und somit das aktualisierte Dienstvorkommen für den Benutzer regelmäßig dargestellt wird.

Als nächstes ruft die *startSearchServices()* die Methode *scanEnvironment()* der *Connection* Klasse auf. Diese Methode wurde bereits oben in dem Vorgang der manuellen Anmeldung beschrieben. Nachdem die Umgebung abgesucht wurde, holt sich die Instanz der Klasse *ScanThread* die Liste der aufgefundenen Server.

Bis hierhin hat sich die Suche auf die in der physikalischen Umgebung befindlichen Dienst-

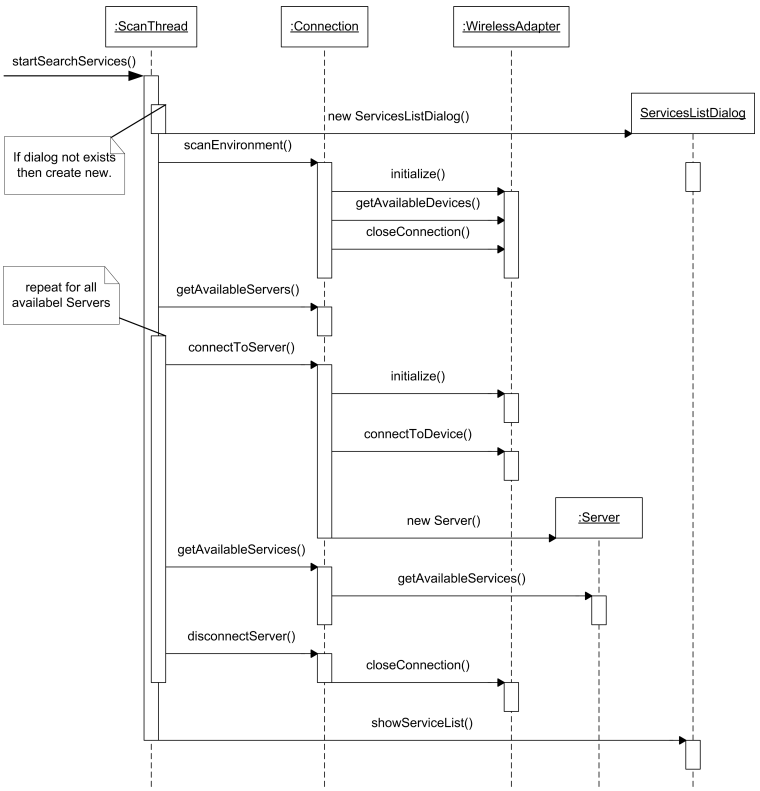


Abbildung 3.7: Sequenzdiagramm: Umgebungsscann auf der Dienstebene.

server konzentriert. An dieser Stelle werden die einzelnen Server nach deren Diensten abgefragt. Hierzu wird eine Iteration über alle vorgefundenen Dienstserver vorgenommen. In dieser wird dann eine Verbindung zu dem jeweiligen Server aufgebaut, alle verfügbaren Informationen geholt und diese in einem neu instantiierten Objekt der Klasse *Server* gespeichert. Diese hält alle wichtigen Informationen des Dienstservers. Alle auf dem Server residierenden Dienste werden dann mit der Methode *getAvailableServices()* geholt. Um so wenig wie möglich Energie der Akkus des PDA zu verbrauchen, wird danach sofort die Verbindung zu dem Dienstserver beendet.

Die Instanz der Klasse *ScanThread* hält an dieser Stelle alle Dienstinformationen aller im Umkreis verfügbaren Dienstserver. Diese Informationen werden dann an die bereits instantiierte Klasse *ServicesListDialog* gesendet. Dieses wird durch den Aufruf der Methode *showServiceList()* gemacht. Hierdurch erscheint in dem Dialog die Liste der verfügbaren Dienste. Der eben geschilderte Vorgang wird noch einmal grafisch mithilfe eines Sequenzendiagramms in der Abbildung 3.7 dargestellt.

### Verbindung mit einen Dienst

Wie in der Beschreibung des Anwendungsfalls *Verbindung mit einen Dienst* in Abschnitt 3.4.1 erwähnt, wird für eine erfolgreiche Verbindung zwischen dem PDA und einen Dienst eine Verbindung mit dem Dienstserver, auf dem der jeweilige Dienst residiert, vorausgesetzt. Dieser Vorgang wurde bereits in dem Abschnitt zuvor erläutert und wird hier nicht weiter erklärt.

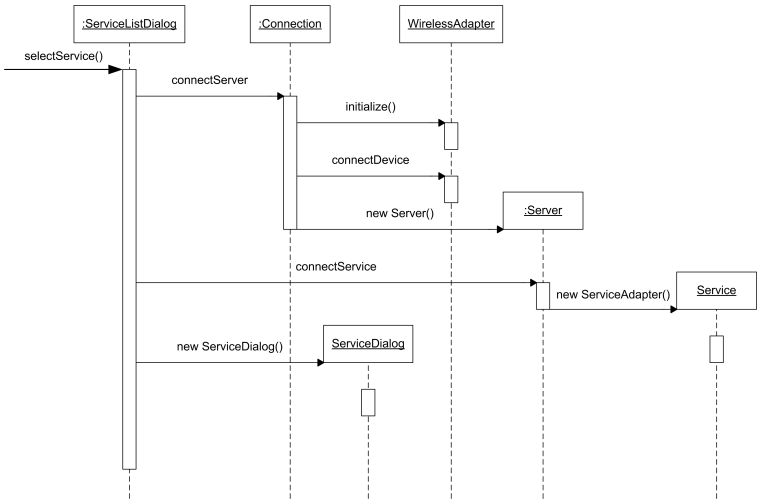


Abbildung 3.8: Sequenzdiagramm: Verbindung mit einen Dienst.

Die Verbindung mit einem Dienst wird von dem Benutzer initiiert. Dieser sieht den ServiceListDialog mit den in der Umgebung vorgefundenen Multimedia Diensten. Von dem kann der Benutzer einen auswählen, wodurch eine Verbindung mit dem Dienst aufgebaut wird. Nachdem also eine Verbindung zu dem Dienstserver aufgebaut wurde, wird die Methode *connectService()* der Klasse *Server* aufgerufen. Diese veranlasst das Serverobjekt eine neue Instanz der Klasse *Service* zu instantiieren, welche als Verbindungsglied zwischen dem eigentlichen Dienst und den Interessenten fungiert. Durch diese werden alle Anfragen an den Dienst gestellt. Durch diese Adapter-Klasse soll eine Möglichkeit geschaffen werden, das Dienstinterface jederzeit zu erweitern, ohne dabei die alten Dienste selbst verändern zu müssen.

Nachdem die Verbindung mit dem Dienst über die Adapter Klasse aufgebaut wurde, instantiiert die *selectService()* Methode ein Objekt der Klasse *ServiceDialog*. Dieser Dialog hat dann als Aufgabe, nach der Verbindung mit dem ausgewählten Dienst, diesen für den Benutzer darzustellen. Der eben geschilderte Vorgang wird noch einmal grafisch indem Sequenzdiagramm 3.8 dargestellt.

### Kommunikationsbehandlung

Die Behandlung der bei der Kommunikation entstehenden Probleme stellt eine große Herausforderung an die mobile Multimedia Middleware dar. Bei der Betrachtung müssen verschiedene Aspekte betrachtet werden, die die Kommunikation stören können. Diese können beliebig komplex sein. Aus diesem Grund wird als Erstes ein Aktivitätsdiagramm zur Verdeutlichung der Problematik und zur Verfeinerung des oben beschriebenen Anwendungsfalls folgen. Für die weitere Klarheit werden darauf folgend die entsprechenden Sequendiagramme dargestellt.

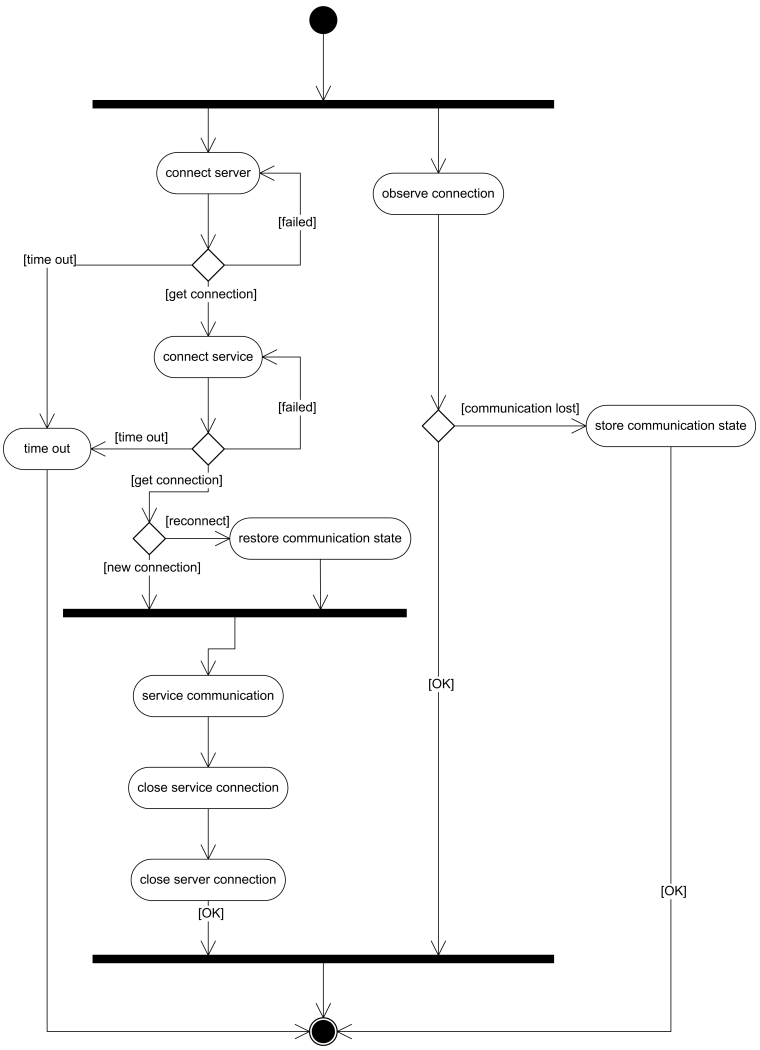


Abbildung 3.9: Aktivitätsdiagramm: Kommunikation sowie die Problembehandlung.

Das Aktivitätsdiagramm 3.9 schildert im Groben, wie die Behandlung des Kommunikationsproblems aussehen kann. Dieser fängt mit dem Aufbau der Kommunikationsverbindung

mit einem Server an. Diese Aktivität wird ggf. mehrmals wiederholt. Wird keine Verbindung aufgebaut, wird weiter zu der Aktivität *time out* verzweigt. Von dort aus werden die entsprechenden Aktivitäten durchgeführt, die den Benutzer und das System über den Timeout informieren. Nach dem Timeout wird der automatisierte Verbindungsaufbau beendet. Dieser kann nur durch den Benutzer neu gestartet werden.

Wird erfolgreich eine Verbindung zu einem Server aufgebaut, so wird weiter zur nächsten Aktivität *connect service* verzweigt. Hinter dieser stehen alle erforderlichen Aufgaben, die notwendig sind, um eine Verbindung mit einem Service aufzubauen. Ähnlich, wie das bei dem Server der Fall war, wird auch hier der Versuch, eine Verbindung aufzubauen, bei Fehlschlag mehrmals wiederholt. Auch hier wird nach einigen Fehlschlägen zu der Aktivität *time out* weiter verzweigt.

Wird eine Verbindung zum einen Dienst aufgebaut, so wird weiter eine Unterscheidung zwischen einem Neuaufbau und einem Wiederaufbau der Verbindung getroffen. Nur in dem Fall, dass der Aufbau der Verbindung ein Wiederaufbauversuch ist, welcher durch einen ungewollten Verlust der Verbindung hervorgerufen wurde, wird zu der Aktivität *restore communication state* verzweigt. Dieser steht für alle notwendigen Aufgaben, die getätigt werden müssen, um den Zustand wiederherzustellen der vor dem Verbindungsverlust bestanden hat. Somit wird den System und den Benutzer die Möglichkeit geben, da anzusetzen, wo die Verbindung unterbrochen wurde. Nach der Herstellung der Umgebung wird weiter zur der Aktivität *service communikation* verzweigt.

Die Aktivität *service communikation* steht in diesem Fall für alle Aktivitäten, die bei einer Kommunikation mit einem Dienst und deren Nutzung anfallen. Wurden alle Aufgaben verarbeitet und ist die Kommunikation mit dem Dienst nicht mehr erforderlich, wird zur der Aktivität *close service connection* verzweigt. In dieser wird die Dienstkommunikation ordnungsgemäß beendet. Ist dieser erfolgreich, so wird in der darauf folgenden *close server communication* Aktivität die Verbindung mit dem Server sauber beendet.

In den meisten Fällen wird der Ablauf wie zuvor geschildert aussehen. In der Praxis ist damit zu rechnen, dass die Verbindung nicht aufrechterhalten werden kann, d.h. gestört wird. Dieses hätte zur Folge, dass die Kommunikation abrupt unterbrochen wird und das System in einem unsicheren Zustand gelassen wird. Des Weiteren werden die bisher erledigten Aufgaben bzw. damit verbundenen Daten verloren gehen. Dieses sollte unterbunden werden. Hierzu wird parallel zu dem Verbindungsaufbau eine weitere Aktivität gestartet. Die Aktivität *observe connection* kapselt alle Aufgaben, die für die Erkennung von Kommunikationsstörungen zuständig sind. Im Normalfall wird diese bei dem Beenden der Kommunikation ebenfalls ohne Weiteres beendet. Im Fehlerfall erkennt diese den Verbindungsverlust und verzweigt zu einer weiteren Aktivität *store comunication state*. Deren Aufgabe ist es den Zustand des Systems zu speichern, so dass dieser später bei neu aufgebauter Kommunikation restauriert werden kann. Nachdem die Sicherung der Daten beendet worden ist, stellt diese das System in einen gültigen Zustand, sodass dieser einen Neuaufbau der Verbindung starten kann.

In dem Sequenzdiagramm 3.10 wird die Problematik der Aufrechterhaltung der Kommunikation verfeinert. Diese besteht einmal aus der Beobachtung der zu kommunizierenden Entitäten. Des Weiteren der Erkennung der dabei möglicherweise entstehenden Verbindungsprobleme und der hierdurch verursachte Verbindungsverlust, welches durch Wiederherstellung dieser zu behandeln ist.

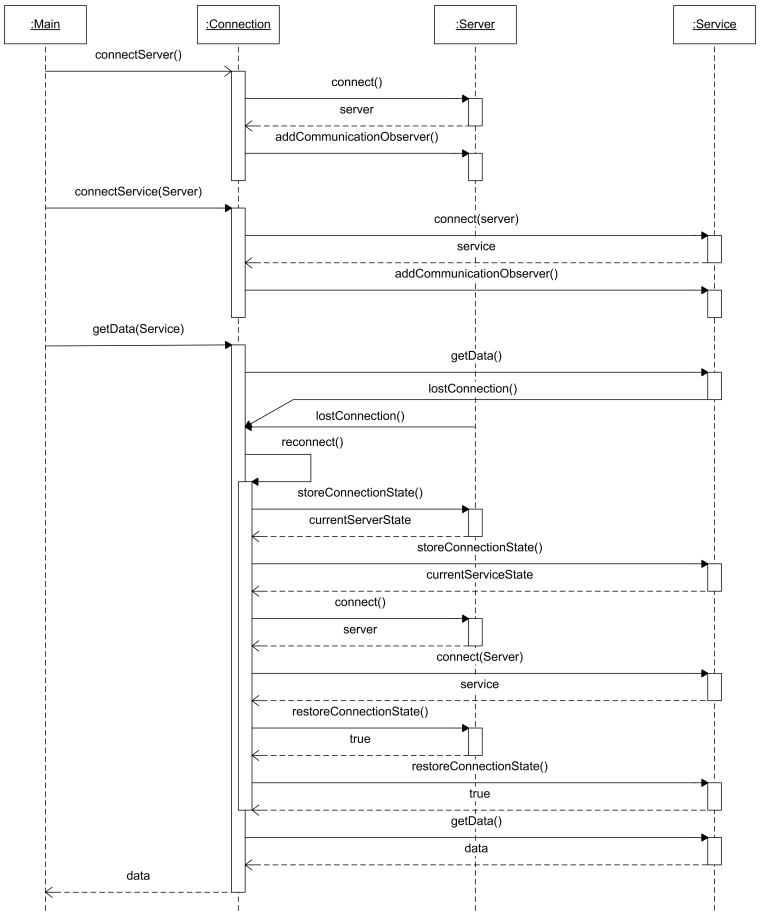


Abbildung 3.10: Sequenzdiagramm: Kommunikation sowie die Problembehandlung.

Bei der Herstellung einer Verbindung mit einem Server sowie einem Dienst werden deren Adapter-Klassen instanziiert und in der Klasse *Connection* aufbewahrt. Dieses wird durch zwei Methoden *connectServer()* und *connectService()* initiiert. Außer der Instanziierung wird in den beiden Methoden eine weitere Methode, die von den beiden Klassen implementiert wird, aufgerufen. Gemeint ist hier die Methode *addCommunicationObserver()*, durch welche die Klasse *Connection* sich als Beobachter der Verbindung bei den beiden Klassen registriert und somit bei Problemen sofort benachrichtigt werden kann.

Als Nächstes wird die Methode *getData(Service)* der *Connection* Klasse aufgerufen, durch welche von dem spezifizierten Dienst Daten gefordert werden. Im Normalfall wird diese Me-

thode ohne Unterbrechung ausgeführt und die weitere Bearbeitung durchgeführt. Wie bereits geschildert worden ist, können bei der Kommunikation Probleme entstehen, sodass diese gestört sein kann. In diesem Fall je nach dem ruft die Klasse *Service* bzw. *Server* die Methode *lostConnection()* der Klasse *Connection* auf, die das ConnectionObserver Interface implementiert. Hierdurch weißt diese, dass die Kommunikation unterbrochen wurde und somit eine Behandlung dieses Fehlerzustandes notwendig ist. Hierzu wird die Methode *reconnect()* aufgerufen.

Die Methode *reconnect()* führt als erstes die Methode *storeConnectionState()* der entsprechenden Klassen auf, wodurch diese ihren aktuellen Zustand sichern, damit dieser später wiederhergestellt werden kann. Danach versucht die Methode *reconnect()* die verlorene Verbindung wiederaufzubauen. Ist dies geglückt, wird durch einen weiteren Aufruf der Methode *restoreConnectionState()* der beiden Klassen *Server* und *Service* der Zustand der Verbindung, so wie dieser vor dem Verbidungsverlust war, wiederhergestellt. War dieses erfolgreich, kann die weitere Kommunikation durchgeführt werden.

#### Dienstadminstration

Der Anwendungsfall 3.11 beruht hauptsächlich auf den bisher besprochenen Anwendungsfällen. So erfordert dieser zum einen, eine Verbindung mit der Umgebung und dem Server. Zum anderen muss sichergestellt werden, dass die Verbindung während der Administrationsaufgaben nicht unterbrochen wird.

Jeder Dienst muss der Umgebung bekannt gegeben werden, bzw. aus dieser entfernt werden. Hierzu sind Administrationsaufgaben notwendig, die bereits in dem Anwendungsfall 3.5 spezifiziert wurden. Hierzu muss sich der Administrator bei der physikalischen Umgebung anmelden. Hierzu wird die Methode *login()* der Adapter-Klasse *Server* aufgerufen. Nachdem die Verbindung hergestellt wurde, wird mit der Methode *registerService()* ein ausgewählter Dienst registriert. Dieses impliziert das Instanziieren der Adapter-Klasse *Service* und somit deren Initialisierung. Ist diese erfolgreich durchlaufen, so kann der neue Dienst mithilfe der Methode *startService()* gestartet werden und somit für andere verfügbar gemacht werden.

Soll aus irgendwelchen Gründen ein bereitgestellter Dienst aus der Umgebung entfernt werden, so muss dieser als erstes gestoppt werden. Dieses wird durch den Aufruf der Methode *stopService* der *Server* Klasse veranlasst. Um bei den Benutzern, die zurzeit diesen Dienst nutzen, einen sauberen Systemzustand zu hinterlassen, meldet die *Service* Klasse alle Benutzer ordnungsgemäß ab. Sind alle Benutzer abgemeldet, kann dieser Dienst aus der Umgebung entfernt werden. Hierzu wird die Methode *deregisterService()* der Klasse *Server* aufgerufen.

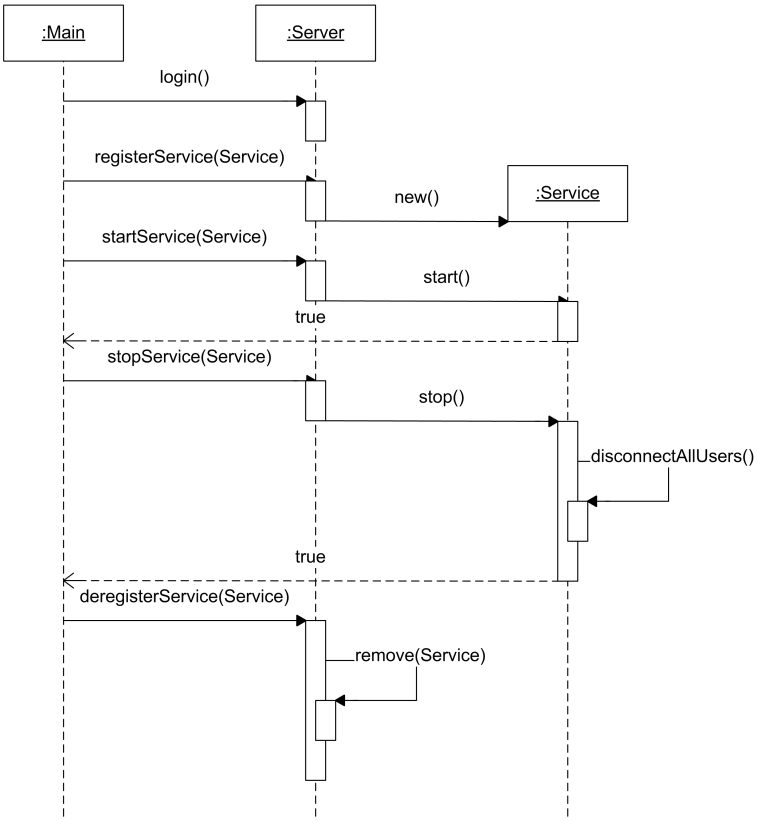


Abbildung 3.11: Sequenzdiagramm: Administration der Dienste.

### 3.4.4 Grobe Architektur

#### Überblick über die Gesammtarchitektur

Um einen Überblick über das hier betrachtete System zu bekommen, wird erstmals ein grobes Gesamtkonzept des Systems definiert und beschrieben. Das System setzt sich aus zwei Kommunikationsseiten zusammen, einen Server und einen Client. Der Server stellt die Multimedia Dienste bereit, der Client greift auf diese zu. Damit stellt das System eine Client/Server Architektur dar. Die jeweiligen Kommunikationsseiten setzen sich aus mehreren Komponenten zusammen, der mobilen Multimedia Middleware, den Multimedia Diensten und der Interaktionskomponente.

Der Übersicht halber wird in dieser Betrachtung die mobile Multimedia Middleware als eine ganze Komponenten angesehen. Diese fungiert in der dienstorientierten Umgebung als eine Kommunikationsmiddleware zwischen den Multimedia Diensten. Aus diesem Grund ist es erforderlich, dass diese auf beiden Kommunikationsseiten installiert wird, damit die erforderliche Infrastruktur auf beiden Seiten gegeben ist. Diese grobe Architektur wird in der Abbildung 3.12 dargestellt.

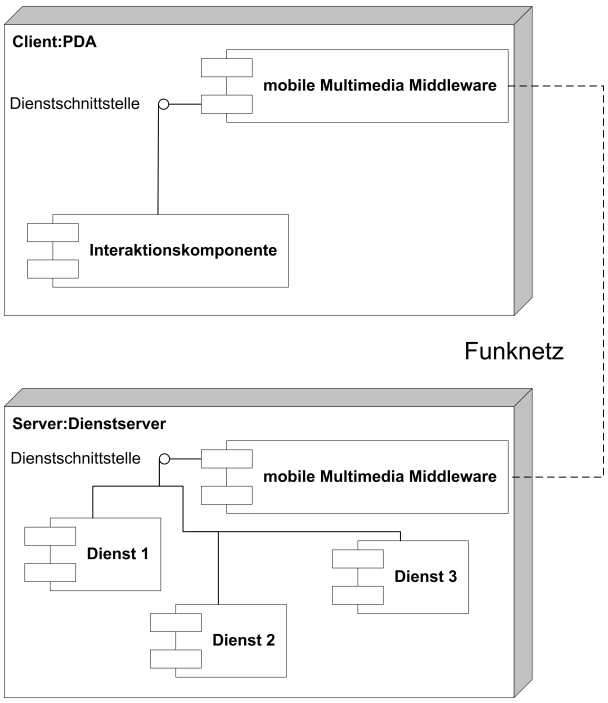


Abbildung 3.12: Grobe Gesammtarchitektur.

Außer den Multimedia Diensten und der mobilen Multimedia Middleware ist in der Abbildung 3.12 ebenfalls eine Zugangskomponente zu erkennen. Diese steht für Clientsoftware die notwendig ist, um auf die bereitgestellten Dienste zugreifen zu können. Im Grunde stellt diese einen weiteren Multimedia Dienst dar, welcher sich geringfügig von den anderen unterscheidet, indem dieser die Darstellungs- und Interaktionsmöglichkeiten der anderen Dienste dem Benutzer anbietet. Diese stellt den Zugang zu den im Umfeld gefundenen Multimedia Diensten und dem Benutzer hier.

### Technologiensichtweise

Der Aufbau der mobile Multimedia Middleware wird im Groben in der Abbildung 3.13 dargestellt. Dabei werden die zum Einsatz kommenden Technologien berücksichtigt.

Als Voraussetzung wird eine Java Virtuelle Maschine (JVM) vorausgesetzt. Diese beinhaltet das standardisierte Wireless Interface, durch welches die propretären Funkmodultreiber von der Anwendung abgekoppelt werden. Durch die Abkopplung ist jederzeit ein Austausch der Treiber möglich, ohne dass dabei ein Einfluss auf die Anwendung ausgeübt wird.

Auf der JVM wird eine Implementierung der CORBA Technologie aufgesetzt. Hierdurch wird die Kommunikation zwischen den Kommunikationspartnern bereitgestellt. Nebenbei bietet der CORBA-Standard weitere Dienste an, die bei der weiteren Betrachtung von Nutzen sein können.

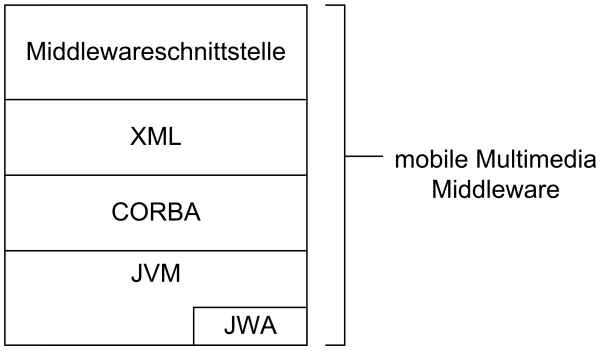


Abbildung 3.13: Grobe Darstellung der mobile Multimedia Schicht.

Das Kommunikationsprotokoll wird mithilfe von XML definiert. Hierdurch gewinnt man eine größere Flexibilität, was die Semantik und die Strukturierung der zu übertragenden Daten angeht. Durch die Angleichung der XML-Handler-Klasse kann das Protokoll jederzeit gezielt erweitert werden.

Ganz oben befindet sich die tatsächliche Anwendungslogik. Diese bildet den Kern, der geschäftsspezifische Funktionalität. Außerdem wird hier die Schnittstelle für die Multimedia Dienste bereitgestellt.

### Vier Schichten Architektur

Der in der Abbildung 3.13 dargestellte grobe Aufbau sollte weiter in eine geeignete Schichtenarchitektur überführt werden. In den meisten Fällen wird eine Drei-Schichten-Architektur verwendet. Die komponentenbasierte und objektorientierte Vier-Schichten-Architektur, die in Steinweg (2002) vorgestellt wird, besitzt aber noch mehr Vorteile.

Diese deckt die im Kapitel 2 gestellten Anforderungen im größeren Maß als die Drei-Schichten-Architektur ab. So stellt diese, außer dem aus der Drei-Schichten-Architektur entstehenden Vorteilen zusätzlich ein weitgehend unabhängiges logisches Design, welches durch die Verwendung von Komponenten, die themenorientierte fachliche bzw. technische Funktionalität durch ein Interface nach Außen kapseln, entsteht. Hierdurch wird der Einsatz der mobile Multimedia Middleware auf heterogenen Zielplattformen begünstigt. Des Weiteren wird durch die Kapselung der Daten und Funktionalität in Komponenten ein Maß an Flexibilität, Wiederverwendbarkeit und Wartbarkeit erzielt. Die Erweiterung der Architektur durch eine vierte Schicht bringt mit sich eine aufgabenbezogene Strukturierung der Komponenten. Hierdurch wird die Portabilität im höheren Masse unterstützt. Die eben geschilderte Architektur wird in der Abbildung 3.14 veranschaulicht.

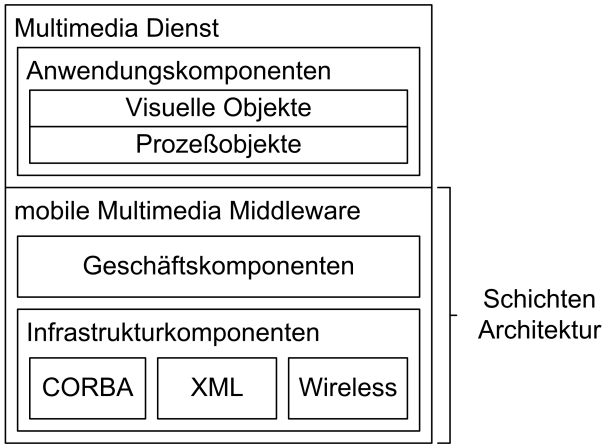


Abbildung 3.14: Mobile Multimedia Middleware aus Sicht der Vier Schichten Architektur.

**Schichtensichtweise**

Bei der Diskussion der Schichtenarchitektur wird in James F. Kurose (2002) auf konzeptionelle Nachteile dieser hingewiesen. In Verhältnis zu dem Vorteilen, die eine Schichtenarchitektur mit sich bringt, haben hier die Nachteile keine große Gewichtung und beeinträchtigen das hier betrachtete System nicht negativ.

Im Abschnitt 3.4.4 wurde die mobile Multimedia Architektur als Vier-Schichten-Architektur identifiziert. Um das System noch flexibler zu machen, wird die mobilen Multimedia Middleware noch weiter in Schichten verfeinert.

Hierdurch eröffnen sich die in (James F. Kurose, 2002) genannten Vorteile. So wird im größten Maße die Offenheit der Architektur unterstützt. Ein bereitgestelltes System, welches Mehrschichtarchitektur aufweist, erlaubt jederzeit die Änderung der einzelnen Schichten. Solange eine Schicht immer noch das gleiche Interface und Funktionalität für die obere Schicht bietet und die Schnittstellen der darunter liegende Schicht benutzt, hat diese Änderung keine negativen Auswirkungen auf das restliche System. Durch diese Tatsache ist die Implementierung der einzelnen Schichten nicht als unveränderbar zu betrachten, vielmehr kann diese je nach Anforderung ausgetauscht werden. So kann das System beliebig nach Anforderungen angepasst werden und somit für verwandte Problembereiche weiterverwendet werden.

Außer der Wiederverwendbarkeit bietet die Schichtenarchitektur eine Trennung zwischen den einzelnen Problembereichen. Durch diese Trennung und die Vorgehensweise, dass eine Schicht *n* die Dienste der Schicht *n*-1 benutzt, wird eine Minderung der Abhängigkeiten zwischen den einzelnen Modulen erreicht. Hierdurch wird unterbunden, dass die Abhängigkeiten zwischen den Modulen kombinatorisch explodieren und somit das System für Wartbarkeit, Veränderbarkeit, Erweiterbarkeit und andere wichtige Vorsätze verschlossen bleibt.

Die mobile Multimedia Middleware setzt sich aus mehreren Schichten zusammen. Jede einzelne kapselt eine Gruppe von Funktionalitäten, die für die jeweilige Schicht zutreffend

ist. Die einzelnen Schichten, sowie ihre Aufgaben sind:

**Kommunikationsschicht** Die Kommunikationsschicht ist für den Aufbau sowie den sauberen Abbau einer physikalischen Verbindung zwischen Client und dem Server zuständig. Die spätere Kommunikation und damit verbundene Aufgaben sind hier ebenfalls angesiedelt.

**Protokollschicht** Die Protokollschicht implementiert entsprechend die protokollrelevanten Anforderungen. D.h., das in dieser Architektur XML-basierte Kommunikationsprotokoll wird entsprechend aufbereitet und die daraus notwendigen Daten heraus geholt bzw. herein gebracht. Die Definition sowie die Validierung der Nachrichten geschieht ebenfalls in dieser Schicht.

**Verbindungsschicht** In der Verbindungsschicht wird die Funktionalität bereitgestellt, die die Multimedia Dienste vor dem Verbindungsaufbau entsprechend konfiguriert. Ist eine Verbindung aufgebaut, überwacht die Verbindungsschicht deren Zustand. Verbindungsprobleme werden von dieser erkannt und das Speichern des aktuellen Zustandes eingeleitet. Bei einer Neuverbindung wird der alte Verbindungszustand wiederhergestellt.

**Sitzungsschicht** Die Sitzungsschicht ist die oberste Schicht in der mobilen Multimedia Middleware. Diese regelt die Problematik der einzelnen Sitzungen, die der Benutzer gleichzeitig öffnen kann. Außerdem stellt diese eine Implementierung für die Automatisierung der Kommunikation bereit, die notwendig ist, um die oben geforderte Mobilität und Transparenz zu ermöglichen. Einer der Aspekte, die häufig weniger betrachtet werden, ist wohl die Sicherheit. Die damit verbundene Logik ist ebenfalls in der Schicht zu finden.


Abbildung 3.15: Grobe Darstellung der mobile Multimedia Middleware Schichten.

Zwischen der mobile Multimedia Middleware und dem Dienst befindet sich als Bindeglied eine standardisierte Schnittstelle, über die die beiden Architekturkomponenten miteinander kommunizieren. Diese Vorkehrung erlaubt es Entwicklern, das Innere der mobile Multimedia Middleware weiter zu entwickeln und die alte Version ablösen. Die Multimedia Dienste können - zumindest theoretisch - weiterhin zusammenarbeiten. Zur Verdeutlichung wird diese abstrakte Schichtenarchitektur bildlich in der Abbildung 3.15 dargestellt.

## 3.5 Systemkomponenten

### 3.5.1 Systemzerlegung

An dieser Stelle sollte die in dem Abschnitt 3.4.4 festgelegte Architektur weiter durch darin enthaltenen Komponenten verfeinert werden. Dabei werden die einzelnen Komponenten nach Anwendungskontroll-, Geschäfts- und Infrastrukturkomponenten unterschieden. Hierdurch werden die einzelnen Aufgabenbereiche noch klarer definiert.

Die gesamte Architektur setzt auf den hier verwendeten Infrastrukturkomponenten auf. Diese werden bei der Modellierung als Black-Boxen betrachtet, welche die Systeminfrastruktur für mobile Multimedia Middleware bereitstellen. Die hier genutzte CORBA-Implementierung sowie die Adapter-Klasse um diese herum wird in diesem Fall als Infrastrukturkomponenten erkannt. Darunter klassifizieren sich ebenfalls die *JVM* und deren Hilfsdienste und Bibliotheken. Der hier zum Einsatz kommende XML-Parser gehört auch der Infrastruktur an. Durch die Black-Box-Modellierung werden diese hier nicht weiter betrachtet.

Die grobe Zerlegung der mobilen Multimedia Middleware in eine Schichtenarchitektur wurde bereits in dem Abschnitt 3.4.4 durchgeführt. An dieser Stelle werden die dort identifizierten Schichten nacheinander einzeln betrachtet und die darin enthaltenen Module identifiziert.

#### Kommunikationsschicht

In der Kommunikationsschicht werden folgende Module identifiziert:

**Dienstauffindung** Das Modul kapselt alle Methoden, die notwendig sind, um Multimedia Dienste in der mobile Multimedia Middleware zu finden. Diese werden unabhängig von den gefundenen Servern in diesem Modul gesucht und protokolliert. Bei der Suche wird nur überprüft, ob der gefundene Dienst auch einem Multimedia Dienst entspricht. Was die Auffindung der eigentlichen Server angeht, wird diese durch die Funkmodule und deren Kommunikationsprotokolle bereitgestellt.

**Verbindungsauf- und Abbau** Dieses Modul kapselt Methoden, die nötig sind, um eine TCP/IP Verbindung zwischen einem Server und einem Client auf- und abbauen zu können. Die Funktionalität, um eine Verbindung zwischen den Funkmodulen aufbauen zu können,

bringen diese selber als Dienst mit. Die Auffindung dieser in der physikalischen Umgebung bringen diese ebenfalls mit.

**Kommunikation** Ist ein gültiger Dienst gefunden und die Verbindung mit diesem Dienst besteht bereits, so kommt es zu der eigentlichen Kommunikation. Dabei werden Nachrichten empfangen und gesendet. Die dazu notwendigen Methoden werden durch dieses Modul bereitgestellt.

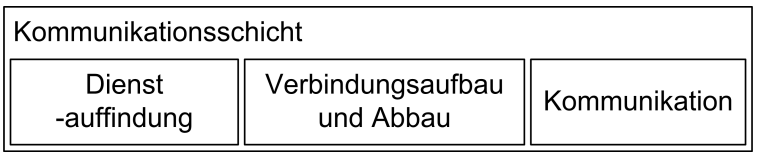


Abbildung 3.16: Komponenten der Kommunikationsschicht.

Die in der Kommunikationsschicht identifizierten Module werden noch einmal bildlich in der Abbildung 3.16 dargestellt.

### Protokollschicht

In der Protokollschicht werden folgende Module identifiziert:

**Nachricht Kodierer** Dienstrelevante Daten können unterschiedlicher Natur sein. Diese können von Dienst zu Dienst in ihrer Semantik und Struktur unterschiedlich aussehen. Um diese vernünftig zu strukturieren und mit einer bestimmter Semantik zuordnen zu können, werden diese in ein XML Dokument kodiert.

**Nachricht Dekodierer** Jede Nachricht, die über die Kommunikationsschicht empfangen wird, ist wie schon erwähnt in einen XML-Dokument gespeichert. Um diese wieder zu gewinnen, wird der Dekodierer bereitgestellt.

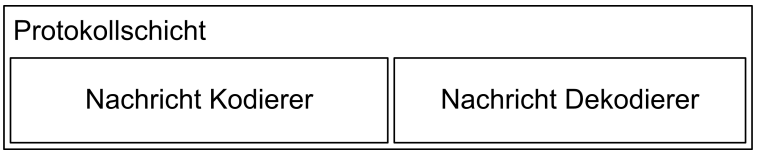


Abbildung 3.17: Komponenten der Protokollschicht.

Die in der Protokollschicht identifizierten Module werden noch einmal bildlich in der Abbildung 3.17 dargestellt.

### Verbindungsschicht

In der Verbindungsschicht werden folgende Module identifiziert:

**Observer** In einer sich ständig ändernden Umgebung treten neue Probleme zum Vorschein. Ein oben identifiziertes Hauptproblem ist wohl die Aufrechterhaltung der Kommunikation. Es ist also erforderlich, die aufgebaute Kommunikation kontinuierlich zu überwachen. Hierzu wird dieses Modul entwickelt.

**Dienstidentifikation** Wie das bei der Web Services Technologie der Fall ist, so werden auch die Multimedia Dienste durch eine variable Anzahl von Eigenschaften beschrieben. Diese Maßnahme erlaubt eine bessere Identifizierung der gesuchten Multimedia Dienste. Dieses Modul stellt hierfür notwendige Funktionalität bereit. Außerdem werden hier verschiedene Filtermethoden für die Suche bestimmt.

**Recovery** Ist ein Kommunikationsproblem aufgetreten, wodurch die Verbindung unterbrochen wurde, so ist es absolut notwendig, bei einem Neuaufbau an der Stelle, kurz vor der Unterbrechung, in der Bearbeitung fortzufahren und somit diese weiter nach oben transparent zu machen. Hierzu ist eine Aufzeichnung des Zustandes der Verbindung notwendig. Außer der Aufzeichnung, ist eine Funktionalität notwendig, die den Zustand der neu aufgebauten Kommunikation durch den alten Zustand ersetzt. Durch die Mitprotokollierung des Verbindungszustandes entsteht zusätzlich ein weiter Vorteil. Das Aufzeichnen stellt im gewissen Sinne einer Pufferung des Verbindungszustandes dar. Hierdurch werden wiederholte Aufrufe sofort beantwortet, ohne zusätzlichen Verbindungsverkehr zu verursachen und somit unnötig Ressourcen zu verschwenden.

**Dienstkonfigurierung** Die Unterstützung der Offenheit gegenüber dem Client erfordert einige Maßnahmen. So muss ein neu gefundener Multimedia Dienst, welcher ausgewählt wurde, entsprechend der Clientvorgaben vor dem Verbindungsaufbau konfiguriert werden, damit der Client und dessen Möglichkeiten im vollen Umfang ausgenutzt werden können. Die hierfür erforderlichen Aufgaben werden in diesem Modul gekapselt.

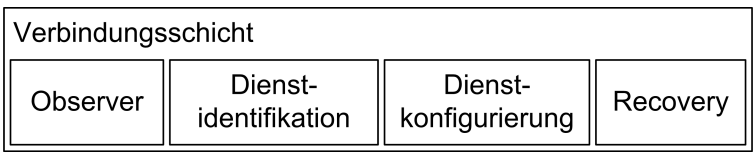


Abbildung 3.18: Komponenten der Verbindungsschicht.

Die in der Verbindungsschicht identifizierten Module werden noch einmal bildlich in der Abbildung 3.18 dargestellt.

#### Sitzungsschicht

In der Sitzungsschicht werden folgende Module identifiziert:

**Dienstkommunikation** Die Kommunikation zwischen Dienst und Client ist auf einer höheren Abstraktionsebene angesiedelt. Für die Regelung der Kommunikation wird die notwendige Funktionalität in diesem Modul gekapselt.

**Dynamischer Auf- und Abbau der Kommunikation** In dieser Architektur ist die Veränderung der Umgebung eine der Vorgaben, die unterstützt werden sollen. Diese zeichnet sich durch die ständige Veränderung der physikalischen Umgebung aus. Dabei findet kontinuierlich das An- und Abmelden bei den Dienstservern statt. Um dem Benutzer den Anschein einer kontinuierlich bestehenden Verbindung geben zu können, wird ein Mechanismus benötigt, der den ständigen Wechsel verbirgt. Im diesem Modul wird die automatisierte Steuerung des Auf- und Abbaus der Kommunikation mit einen Dienst implementiert.

**Sicherheit** Das Modul Sicherheit kapselt alle Aufgaben, die nötigt sind, um eine gegenüber Unbefugten gesicherte Kommunikation zwischen dem Dienst und dem Server aufzubauen.

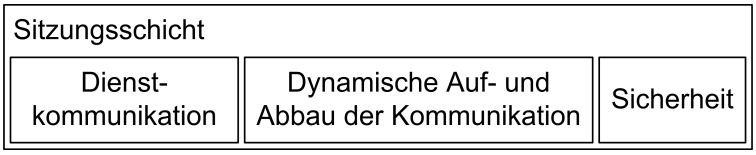


Abbildung 3.19: Komponenten der Sitzungsschicht.

Die in der Sitzungsschicht identifizierten Module werden noch einmal bildlich in der Abbildung 3.19 dargestellt.

#### Middlewareschnittstelle

Die Middlewareschnittstelle ist die Stelle, wo der Entwickler den Aufbau der Dienste (Dienstsyntax, Semantik) definiert. Dies ist Analog zu den Skeletons/Stubst von CORBA zu sehen. Diese Schicht packt die Dienstleistungen in das XML-Dokument, bzw. auf der anderen Seite entpackt sie diese und stellt sie als eine Collection dar. Diese wird dann von Clientprogramm genutzt.

### 3.5.2 Beziehungen

Bei der hier gewählten Schichtenarchitektur sind die Beziehungen zwischen den einzelnen Modulen sehr einfach definiert. So stellt jede Schicht eine Schnittstelle für die nächst höhere

Schicht bereit und greift selbst nur auf die direkt darunter liegende Schicht zu. Über der obersten Schicht befindet sich die Middlewareschnittstelle, auf die ein Dienst zugreifen kann.

## 3.6 Festlegung der Schnittstellen

Im Abschnitt 3.5 wurden die in dem System enthaltenen Systemkomponenten identifiziert und beschrieben. Diese werden durch die verschiedenen Schichten gekapselt. Die, wie im Abschnitt 3.5.2 erwähnt, nur mit den darunter liegenden Schichten kommunizieren. Die dabei von jeder Schicht bereitgestellten Dienste werden in diesem Abschnitt identifiziert und beschrieben.

### 3.6.1 Kommunikationsschicht

| **Dienst** | **Beschreibung** | **Parameter** |
|---|---|---|
| searchServer | Gibt eine Liste der verfügbaren Server zurück. | keine |
| connectServer | Baut eine Serververbindung über ein Funkmodul auf. | Server |
| disconnectServer | Baut eine bestehende Serververbindung ab. | Server |
| searchValideService | Gibt eine Liste der verfügbaren Dienste zurück. Diese werden zuvor nach der Kompatibilität mit der MMM überprüft. | Server |
| send | Sendet das spezifizierte XML-Dokument in dem die Dienstdaten enthalten sind. | Service, XML-Document |
| receive | Empfängt ein XML-Dokument. Dieses wird als Rückgabewert übergeben. | keine |

Tabelle 3.1: Dienste der Kommunikationsschicht

Mithilfe der Methode *searchServer()* kann die physikalische Umgebung nach geeigneten Zugangsservern durchsucht werden. Als Rückgabewert wird eine Liste mit Servern zurückgegeben. Wurden keine Zugangsserver gefunden, so wird eine leere Liste zurückgegeben. Um eine Verbindung mit einem Zugangsserver aufzubauen, muss die Methode *connectServer()* aufgerufen werden. Hierzu muss in dem Methodenparameter der Server spezifiziert werden. Wird die Methode erfolgreich ausgeführt, so wird der initialisierte Server-Adapter zurückgegeben, durch welchen die Kommunikation mit dem Server durchgeführt werden kann. Jetzt kann der Server mit der Methode *searchValidService()* nach kompatiblen Diensten durchsucht werden. Für die Kommunikation stellt die Kommunikationsschicht zwei Methoden bereit. Die Methode *send()* hat die Aufgabe ein XML-Dokument an einen spezifizierten

Multimedia Dienst zu senden. Mit der Methode *receive()* kann dann ein neues, von einem Dienst gesendetes XML-Dokument empfangen werden. Wird die Kommunikation mit dem Server nicht mehr benötigt, wird diese durch die Methode *disconnectServer()* beendet, wobei der gemeinte Server in dem Methodenparameter spezifiziert werden muss. In der Tabelle 3.1 werden die einzelnen Dienste, die die Kommunikationsschicht anbietet, noch einmal aufgelistet.

### 3.6.2 Protokollschicht

| **Dienst** | **Beschreibung** | **Parameter** |
|---|---|---|
| readData | Liest die Daten aus dem XML-Dokument. | Data |
| writeData | Schreibt die Daten in das XML-Dokument. | Data |

Tabelle 3.2: Dienste der Protokollschicht

Die Protokollschicht stellt zwei Methoden bereit. Die *readData()* Methode holt die relevanten Daten aus einem XML-Dokument heraus. Die Methode *writeData()* schreibt die entsprechenden Daten in das XML-Dokument rein. In der Tabelle 3.2 werden die einzelnen Dienste, die die Protokollschicht anbietet noch einmal aufgelistet.

### 3.6.3 Verbindungsschicht

| **Dienst** | **Beschreibung** | **Parameter** |
|---|---|---|
| searchService | Gibt eine Liste der gefundenen Dienste als Rückgabewert zurück. Die Suche nach dem Diensten wird durch eine Reihe von Diensteigenschaften beschränkt. | Properties |
| connectService | Baut eine Verbindung zu einen Dienst auf. | Service |
| disconnectService | Baut eine bestehende Dienstverbindung wieder ab. | Service |
| observe | Fügt eine Serververbindung zu dem Beobachter hinzu, sodass dieser die Verbindung überwachen kann. | Connection |
| reconnectService | Baut eine Dienstverbidung wieder auf. Der Zustand der Verbindung wird ebenfalls wiederhergestellt. | Service |

Tabelle 3.3: Dienste der Verbindungschicht

Durch die Methode *searchService()* stellt die Verbindungsschicht eine Möglichkeit bereit, nach den Multimedia Diensten zu suchen. Dabei werden in dem Methodenparameter die einzelnen Diensteigenschaften spezifiziert, die der gesuchte Dienst erfühlen soll. Als Rückgabeparameter werden die gefundenen Multimedia Dienste zurückgegeben. Um sich mit einem Multimedia Dienst zu verbinden, wird die Methode *connectService()* aufgerufen. Diese gibt dann nach einer erfolgreichen Verbindung den Service-Adapter zurück. Durch den Aufruf der Methode *observe()* und der Spezifizierung der gemeinten Verbindung, kann der Zustand der Kommunikation überwacht werden. Bei einer Kommunikationsunterbrechung kann die Methode *reconnectService()* aufgerufen werden, wodurch die Verbindung neu aufgebaut und wiederhergestellt wird. In der Tabelle 3.3 werden die einzelnen Dienste, die die Verbindungsschicht anbietet noch einmal aufgelistet.

### 3.6.4 Sitzungsschicht

| **Dienst** | **Beschreibung** | **Parameter** |
|---|---|---|
| autoConnection | Baut eine automatisierte Verbindung mit dem Dienst auf. | Service |
| getSecureConnection | Baut eine sichere Verbindung mit einen Dienst auf. | Service |
| send | Sendet Daten an einen Dienst. | Server; Data |
| receive | Empfängt Daten von einen Dienst. | Server |

Tabelle 3.4: Dienste der Sitzungschicht

Die Methode *autoConnection()* stellt die notwendige Funktionalität bereit, durch die eine Verbindung mit dem spezifizierten Multimedia Dienst automatisch aufgebaut und aufrechterhalten werden kann. Durch die Methode *getSecureConnection()* wird eine gesicherte Verbindung zu dem spezifizierten Multimedia Dienst aufgebaut. Besteht eine Verbindung mit einem Dienst, so kann die Kommunikation mit dem Dienst mit den Methoden *send()* und *receive()* durchgeführt werden. Dabei werden die dienstrelevanten Daten versendet. In der Tabelle 3.4 werden die einzelnen Dienste, die die Verbindungsschicht anbietet noch einmal aufgelistet.

### 3.6.5 Middlewareschnittstelle

Das Protokoll der Middleware definiert, wie die Dienste einander Nachrichten zuspielen. Insbesondere definiert das Protokoll die ausgetauschten Nachrichten, die Syntax der Nachrichttypen, die Semantik der enthaltenen Felder und Regeln für das Versenden und Empfangen der Nachrichten. Die Methode *loadMethodes()* übergibt alle Methoden eines Multimedia Dienstes als eine Liste zurück. Die Methode *writeMethodes()* schreibt alle Methoden, die ein Multimedia Dienst bereitstellt. Mit der Methode *executeMethode()* kann eine in dem

| **Dienst** | **Beschreibung** | **Parameter** |
|---|---|---|
| loadMethodes | Liest alle verfügbaren Dienstmethoden, die ein Dienst bereitstellt, aus und gibt diese als Liste zurück. | keine |
| writeMethodes | Schreibt alle bereitgestellten Dienstmethoden. | ServiceList |
| executeMethode | Führt Spezifizierte Dienstmethode aus und gibt das Ergebebniss zurück. | ServiceMethode |

Tabelle 3.5: Dienste der Middlewareschnittstelle

Methodenparameter definierte Methode ausgeführt werden. In der Tabelle 3.5 werden die einzelnen Dienste, die die Middlewareschnittstelle anbietet, noch einmal aufgelistet.

## 3.7 Dienstkommunikation

Der Kommunikationsverlauf ist vergleichbar mit dem, welcher bei SOAP-Technologie vorzufinden ist. Wie bereits beschrieben, werden bei SOAP und somit auch bei Web Services, hauptsächlich die Internetprotokolle verwendet. Ein Protokoll, das sich in diesem Umfeld schon seit langem bewährt hat, ist das HTTP[9]. Dieses definiert die Struktur und den Austausch von Nachrichten. SOAP-Technologie nutzt das HTTP um die SOAP-Nachrichten, die XML-Dokumente darstellen, zwischen den beiden Kommunikationsseiten zu transportieren. Bei der mobilen Multimedia Middleware wird eine ähnliche Vorgehensweise verfolgt.

Zwischen dem Client und dem Server werden XML-Nachrichten versendet. Diese enthalten alle erforderlichen Informationen um eine Anfrage an einen Multimedia Dienst stellen zu können. Auf der anderen Seite bietet dieser auch die Möglichkeit die Antwortdaten an dem Client zu liefern, der die Anfrage gestellt hat. Die Abbildung 3.20 skizziert grob den Kommunikationsverlauf.

Außer den Dienstdaten werden weitere mitversendet. Um die Kommunikation zwischen mehreren Clients und Diensten koordinieren zu können, müssen zusätzliche Infrastrukturdaten versendet werden. Hierdurch wird die Regelung und die richtige Zuordnung der physikalischen sowie logischen Kommunikationskomponenten erst ermöglicht.

Aus der Sicht der mobilen Multimedia Middleware sind die Infrastrukturinformationen fundamental. Diese sind bei der Bereitstellung der Mobilität unabdingbar. So werden dabei folgenden Informationen übertragen:

- Sitzungsinformationen
- Umgebungsinformationen

[9] Hypertext Transfer Protokol

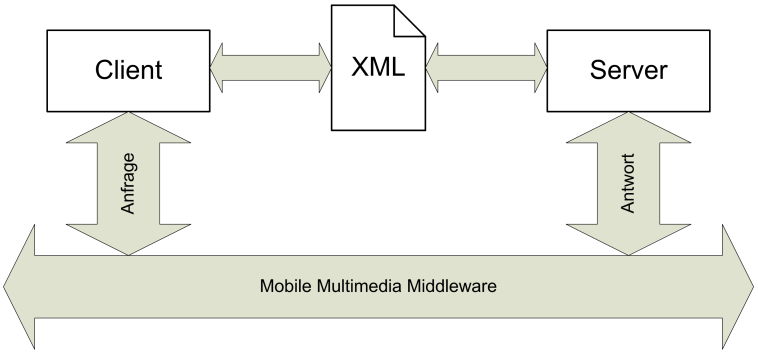


Abbildung 3.20: Kommunikationsverlauf.

- Diestzustandsinformationen

Bei den Sitzungsinformationen handelt es sich um Daten, die für das Sitzungsmanagement notwendig sind. Darunter fallen Verbindungsstatusmeldungen und für die Indentifizierung der Sitzung notwendigen Informationen.

Für die erweiterte Mobilität werden außer den Sitzungsinformationen die Umgebungsinformationen benötigt, durch welche die physikalische Umgebung beschrieben wird. Hierdurch soll der Umgebungswechsel abgewickelt werden.

Ebenfalls ist der Zustand des Multimedia Dienstes wichtig. Dieser Dienst könnte ausgefallen, ausgelastet oder in einer nicht kompatiblen Version vorhanden sein. Durch die Übermittlung der Zustandsinformationen wird es möglich, auf diese Gegebenheiten entsprechend reagieren zu können.

Zusätzlich zu den bereits genannten Aufgaben, die das Kommunikationsprotokoll hat, kommt eine Weitere hinzu. In der Anfangsphase der Client/Dienst-Interaktion ist es notwendig den Client über die auf dem Server residierenden Dienste zu unterrichten. Durch die Möglichkeiten, die XML bietet, wird eine den gestellten Ansprüchen gerechte Dienstbeschreibung mitgeliefert.

## 3.8 Test und Verifikation des Systems

Bis hierher wurde der Entwurf der mobilen Multimedia Middleware vorgestellt. Hierdurch wird verdeutlicht wie das System aussehen soll, aber nicht wie es auf seine Funktionsfähigkeit überprüft werden kann. Mit Funktionsfähigkeit ist an dieser Stelle die Erfüllung der Anforderungen gemeint, die an das System gestellt wurden.

In diesem Abschnitt werden Testszenarien beschrieben, die die Erfüllung, der an die mobilen Multimedia Middleware gestellten Anforderungen, bestätigen. Damit ist also die Bereitstellung einer Infrastruktur gemeint, die die Mobilität dem darauf aufbauenden Clients

ermöglicht. Im Anschluss werden dann Indizien genannt, durch die man auf die Erfüllung der Anforderungen zurück schließen kann.

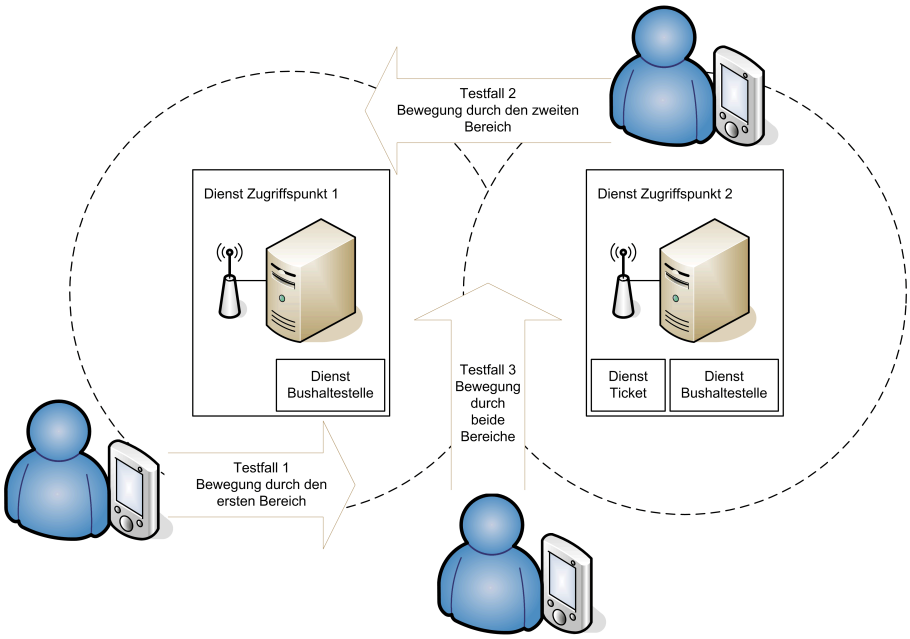


Abbildung 3.21: Physikalische Testumgebung.

Die Abbildung 3.21 veranschaulicht die Testfälle, welche die Funktionsfähigkeit des hier entwickelten Systems testen sollen. Diese untersuchen unterschiedliche Aspekte des Systems und sollten deswegen im Folgenden näher beschrieben werden:

1. Als Erstes wird das Eindringen in die Reichweite eines Servers bzw. seines Bluetooth Hotspots getestet. Ist das geschehen, wird das Durchsuchen nach verfügbaren Multimedia Diensten durchgeführt. Mit den gefundenen Dienstprototypen wird dann eine Verbindung aufgebaut und anschließend dessen Dienstleistungen benutzt. Dabei wird nur ein Dienst zur gleichen Zeit ausgeführt. In diesem Fall wird das System auf die Mobilität in ihrer einfachsten Form geprüft.

2. In diesem Fall wird die Mobilität stärker geprüft. Dabei soll der Client sich aus einem Hotspot Bereich des einen Servers wegbewegen, wodurch eine Verbindungsunterbrechung mit dem jeweiligen Multimedia Dienst erfolgt. Nach dem Verlassen des ersten Hotspot Bereiches soll sich der Client unverzüglich zu dem nächsten Hotspot bewegen. Befindet sich dieser wieder innerhalb eines Hotspot Bereiches, so versucht der Client den Wiederaufbau der Verbindung zu dem Dienst, bei dem die Verbindung unterbrochen wurde. Dabei wird der automatisierte Wiederaufbau getestet.

3. Im diesem Testfall soll der Client durch zwei sich überschneidende Hotspot Bereiche bewegen. Dabei soll die Handhabung der zur Auswahl stehenden Dienste getestet

werden. An dieser Stelle kann z.B. ein Mechanismus getestet werden, welches Benutzerprofile auswertet.

4. Im dem letztem Testfall soll die Wiederaufsetzung auf einen Verbindungszustand getestet werden. Mit anderen Worten, soll hier der Mechanismus überprüft werden, welcher nach einem ungewollten Verbindungsabbruch den Zustand der Verbindung sichert und nach einem erneuten Verbindungsaufbau wieder herstellt, sodass man an der alten Stelle wieder weiter mit der jeweiligen Aufgabe fortfahren kann.

Die ersten drei Testfälle sind in der Abbildung 3.21 durch die Bewegungspfeile dargestellt und nummeriert. Der letzte Testfall wird durch das Abschalten des Zugangspunktes künstlich provoziert.

Durch die Anwendung der Testfälle wird folgendes festgehalten: In dem ersten Testszenario wird die Mobilität in ihrer Grundform überprüft. Diese beinhalten die spontane Vernetzung, Identifizierung der Umgebung und das Anzeigen der dort vorgefundenen Multimedia Dienste.

Im dem zweiten Testfall wird die Überprüfung um ein weiteren wichtigen Aspekt erweitert, wodurch das System auf die erweiterte Mobilität hin untersucht werden kann. Gemeint ist dabei der automatisierte Wiederaufbau einer ungewollt unterbrochenen Verbindung. Dieses wird bei Clients eintreffen die verstärkt ihre physikalische Umgebung wechseln. Im Einzelnen werden hierbei die Erkennung der ungewollten Verbindungsunterbrechung, Wiederauffindung äquivalenten Multimedia Dienstes und der Wiederaufbau der Verbindung untersucht.

In dem dritten Testfall sollte die Handhabung eines neu entstehenden Problems kontrolliert werden. Gemeint ist die Überlappung der Hotspot Bereiche.

Der letzte Testfall hat als Aufgabe verstärkt die Restaurierung des Verbindungszustandes zu untersuchen. Hierdurch sollte das Aufsetzen auf dem Zustand, unmittelbar vor der Verbindungsunterbrechung, überprüft werden.

# Kapitel 4

# Realisierung

## 4.1 Einleitung

Dieses Kapitel beleuchtet die Einzelheiten, die bei der Implementierung des im Kapitel 3 vorgestellten Softwareentwurf beachtet wurden. So befasst sich der erste Abschnitt mit der Hardware und Software, die für die Realisierung de Systems benutzt und somit vorausgesetzt werden. Im Einzelnen werden die Möglichkeiten kurz genannt und nach den für die Arbeit relevanten Gesichtspunkten ausgewählt.

Des Weiteren wird auf die, für die Betrachtung gezogenen, Hardware und Software Komponenten näher eingegangen. Die Beschreibung dient nur dem groben Verständnis und geht nicht ins einzelne Detail. Ist der Informationsbedarf mit diesem Abschnitt nicht gedeckt, so wird der Leser auf weiterführende Literatur verwiesen.

Nachdem die Laborumgebung feststeht, wird zu der Implementierung des Systems übergegangen. Dabei werden die Einzelheiten, die bei der prototypischen Fertigstellung des Systems beachtet wurden, näher erklärt. Dieses sollte dem besseren Verständnis und Motivation der gewählten Implementierung dienen.

## 4.2 Laborumgebung

### 4.2.1 Betrachtung der Testumgebungen

#### Hardware

Wie bereits besprochen, stellt das hier betrachtete System eine Client/Server Architektur dar. Hierzu werden zwei Rechner benötigt. Der Server ist für die Bereitstellung der Dienste zuständig. Des Weiteren muss der Server eine Kommunikationsmöglichkeit bereitstellen, mit der eine TCP/IP-Verbindung mit einem Client aufgebaut werden kann. Darüber hinaus muss die Kommunikation auf einer Funktechnologie basieren.

Bei dem Client-Rechner sehen die Anforderungen etwas anders aus. Dieser darf kein stationärer Rechner sein. Vielmehr muss hier ein kleines, handliches Gerät gewählt werden, welches durch seine Größe und andere technische Anforderung die physikalische Mobilität

eines Benutzers nicht einschränkt. Darunter fallen solche Geräte wie PDA oder Vergleichbares. Notebooks scheiden hier aus, da diese zwar portabel[1] sind, die Mobilität im eigentlichen Sinne nicht wiedergeben. Zusätzlich muss das Clientgerät eine Möglichkeit bieten können, eine TCP/IP-Verbindung über einen Funkmodul aufbauen zu können.

#### Software

In dem Kapitel 2 wurden verschieden Technologien betrachtet, die für die Realisierung verwendet werden können. Dabei wurden ebenfalls die Programmiersprache, sowie die Programmierumgebung gewählt, die J2EE. So ist eine Anforderung, die Unterstützung der Java Plattform. Diese muss sowohl auf dem Client, als auch auf dem Server vorhanden sein, damit das System lauffähig ist.

Außer der Java Plattform wird eine CORBA Implementierung benötigt. Diese muss im Stande sein, nicht nur auf einem herkömmlichen PC zu laufen, sondern auch auf einen PDA. So muss diese Implementierung auch mit Plattformen umgehen können, die nicht so leistungsstark sind, wie vergleichbare PC. Mit anderen Worten muss die Implementierung so schlank wie möglich sein, dabei aber die Grundfunktionalitäten weiterhin unterstützen.

### 4.2.2 Auswahl und nähere Beschreibung

#### Hardware

In der Testumgebung wurde für den Server ein handelsüblicher Desktop PC gewählt. Dieser stellt USB-Ports bereit, an denen das notwendige Funkmodul angeschlossen werden kann.

Das Funkmodul wird in diesem Fall das *TDK Bluetooth USB Adaptor* sein. Dieses erlaubt eine Bluetooth Funkkommunikation über eine Reichweite von bis zu 50 Metern und eine Datenübertragungsgeschwindigkeit von bis zu 721 kbps. Dieses arbeitet mit allen bekannten Bluetooth-Geräten zusammen. Der Bluetooth-Adapter fungiert als Master und ist im Stande bis zu sieben Slaves zu verwalten und somit ein Piconetz[2] aufzubauen. Das Gerät wird in der Abbildung 4.1 dargestellt.

Wie oben bereits genannt, wird für den Client ein PDA genommen. Dieser soll für das Informationssystem als ein mobiles Frontend zum Einsatz kommen, welches eine Funkschnittstelle besitzt, über die die Anbindung an das Informationssystem realisiert werden kann.

---

[1] Portable Geräte sind solche, die von einer Umgebung zu anderen bewegt werden können und in diese integriert werden können. Mobilität erweitert die Portabilität. Ein mobiles Gerät erlaubt zusätzlich, während des Wechsels der Umgebung die Kommunikation mit dem Netzwerk beizubehalten und somit weiter dessen Nutzung.

[2] Mit einander kommunizierende Bluetoothgeräte bilden ein Piconetz. Innerhalb des Netzes fungiert ein Bluetoothgerät als Master. Dieser hat als Aufgabe, die Kommunikation zwischen den restlichen Bluetoothgeräten, auch Slaves genannt, zu koordinieren. Da die Slaves nur über den Master miteinander kommunizieren können, wird die Größe der Piconetzteilnehmer auf acht beschränkt.


Abbildung 4.1: TDK Bluetooth USB Adaptor (TDKSystems, 2004).

Ein weiterer Vorteil, der bei der Verwendung eines PDA entsteht ist, dass im Gegensatz zu den telefonartigen Geräten der PDA in der Regel ein wesentlich größeres Display anbietet, welches die Darstellung der verfügbaren Dienste erleichtert. Die Eingaben werden über einen Touchscreen getätigt. Zusätzlich bietet dieser verglichen mit Funktelefonen eine größere Rechenleistung an, welche notwendig ist, um die eingesetzte Software benutzen zu können.

Einen derartigen PDA bietet die Firma Compaq/HP an. Dieser erfüllt alle an den Client gestellten Anforderungen. Zusätzlich hat dieses Gerät einen Bluetooth-Chip standardmäßig eingebaut, wodurch die Funkkommunikation mit dem Server aufgebaut werden kann. Der hier gewählte iPAQ H3870 ist mit einen 206 MHz Strongarm Prozessor, einen 3,8 Zoll Display, 64 MB RAM, 32 MB ROM ausgestattet. Außerdem bring dieser eine Akku Laufzeit von 14 Stunden mit, sodass dieses Gerät für den Einsatz in einer veränderlichen Umgebung geeignet ist.

Wie schon zuvor erwähnt, ist in dem PDA ein Bluetooth-Funkmodul integriert. Vorteil der Bluetooth-Technologie ist, dass es praktisch im jeden Gerät integriert werden kann. Außerdem ermöglicht diese eine Funkkommunikation mit anderen, innerhalb von 10 Metern befindlichen Bluetooth Modulen. Ein einzelnes Modul kann bis zu sieben andere Module ansprechen und somit ein Nahbereichfunknetz, kurz PAN[3], aufbauen. Mithilfe diese Technologie werden verschiedene Geräte, in diesem Fall ein PDA, befähigt, sich miteinander ohne aufwendige Konfiguration spontan zu vernetzen (Ad-Hoc-Networking). Hierdurch kann die geforderte Spontanität zum größten Teil gewährleistet werden. Dieses gilt aber nur auf der Verbindungsschicht. Weitere Einzelheiten über die Bluetooth Technologie sind aus der

[3]Personal Area Network.

Literatur zu entnehmen (s.z.B. Babic, 2003).


Abbildung 4.2: iPAQ H3870. Quelle:HP (2004)

Zu dem in dem iPAQ H3870 eingebauten Bluetooth Funkmodul ist noch anzumerken, dass dieser einen Sparmodus bereitstellt, sodass dieser die Sendeleistung an die tatsächliche Entfernung zu dem Kommunikationspartner anpasst und somit für längere Laufzeit sorgt. Die Abbildung 4.2 zeigt das eben beschriebene Gerät.

### Software

**Client** Das auf dem PDA laufende Betriebssystem ist das von Microsoft für mobile Geräte entwickelte *PocketPC 2002*[4]. Dieser beinhaltet einen Bluetooth-Manager, mit dessen Hilfe man das in dem Gerät eingebaute Bluetooth-Funkmodul bedienen kann. So kann dieser das Bluetooth-Funkmodul einschalten und dazu auffordern, die nähere Umgebung nach anderen Bluetoothgeräten zu durchsuchen, mit denen eine Funkkommunikation aufgebaut werden kann. Wird ein solches Gerät gefunden, so kann über den Bluetooth-Manager die Kommunikation mit diesem aufgebaut und später beendet werden.

Wie bereits besprochen, wird *Java* als Programmiersprache verwendet. Um die hier entwickelte Anwendung auch auf dem Client PDA verwenden zu können, ist es notwendig auf dem PDA auch eine Java Runtime Environment (JRE) zu installieren. Der hier ausgewählte PDA wird zusammen mit der JRE Implementierung *JeodeTM PDA Edition* der Firma Insi-

[4] http://www.microsoft.com/windowsmobile/products/pocketpc/default.mspx

gnia mitgeliefert. Diese entspricht der *PersonalJava 1.2*[5] und *EmbeddedJava*[6] Spezifikation und ist somit äquivalent zu Java 1.1.8. Diese wurde speziell für mobile Geräte, wie PDA und Handheldcomputer entwickelt. Die Einzelheiten sind aus (Insignia, 2004) zu entnehmen.

Die hier entwickelnde Software benötigt weitere Hilfssoftware. So wird für die Kommunikation eine CORBA Implementierung namens *JacORB 1.3.30*[7] und XML Prozessor Implementation *JAXP 1.2.4* benutzt. JacORB ist eine Implementierung des CORBA 2.0 Standards und eine reine Java Anwendung, welche in der eingeschränkten JRE ausgeführt werden kann. Außer dass der Leistungsumfang der Implementierung alle notwendigen Anforderungen abdeckt, die hier an die CORBA Implementierung gestellt wurden, ist JacORB frei verfügbar, sodass dieser ORB für den Einsatz hier die beste Wahl darstellt. Für weitere Information wird auf (JacORB, 2004) verwiesen.

Wie bereits gesagt, stellt *JAXP* einen XML-Prozessor bereit, wodurch Anwendungen XML-Dokumente parsen und transformieren können. Durch *JAXP* spezifizierte Schnittstelle ist die Benutzung der Parser von der Implementierung entkoppelt und kann somit frei gewählt werden. Weiter Informationen sind unter (JAXP, 2004) zu finden.

**Server** Auf dem Desktop PC, welcher als Dienstserver fungiert, wird als Betriebssystem Microsoft Windows XP[8] verwendet. Diese erfüllt alle notwendigen Anforderungen für die Fertigstellung der hier bearbeiteten Aufgabe.

Zusätzlich hierzu wird auf dem Rechner das JRE 1.4.2 installiert. Diese ist so wie bei dem Client notwendig, um die hier entwickelte Anwendung ausführen zu können. So wie beim Client der Fall ist, so wird auch auf dem Server die Hilfssoftware *JacORB* und *JAXP* verwendet.

**Entwicklungsplattform** Die eben vorgestellten beiden Rechner stellen die Testumgebung dar. Für die Entwicklung der Anwendung wurde ein mit Server vergleichbarer Rechner verwendet. Auf diesem wurde so wie bei dem Server genannte Software installiert.

Zusätzlich hierzu wird ein *PersonalJava 2.1* Emulator installiert, welcher für die Prüfung der Konformität der entwickelten Software mit der Java 1.1.8 benutzt wird. Die Implementierung des Emulators wird ebenfalls von Sun bereitgestellt. Weiter führende Informationen findet man unter (PersonalJava, 2004). Eine weitere Software, die für die Unterstützung der Entwicklung auf dem Rechner installiert wurde, ist *Eclipse*. Diese stellt eine vollwertige Java Entwicklungsumgebung bereit und ist frei verfügbar. Die Einzelheiten sind aus (Eclipse, 2004) zu entnehmen.

---

[5] inzwischen durch J2ME Personal Profile abgelöst

[6] inzwischen durch J2MECDC/Foundation Profile abgelöst

[7] JacORB ist eine von der Freien Universität Berlin, Software Engineering and Systems Software Group and Xtradyne Technologies AG gemeinsam entwickelten CORBA Implementierung.

[8] http://www.microsoft.com/windowsxp/default.asp

## 4.3 Konzeptumsetzung

### 4.3.1 Umsetzung der Komponenten

An dieser Stelle sollte die Umsetzung der Komponenten betrachtet werden. Diese wird eher auf der konzeptionellen Ebene angesiedelt sein und betrachtet weniger die Einzelheiten der Implementierung. So werden in der Realisierung Entwurfsmuster verwendeten, die ausführlich in Gamma u. a. (1996) beschrieben werden.

Bei der Realisierung der Middlewareschnittstelle, welche das Bindeglied zwischen einem Dienst und einen Server ist, wird das *Fassaden*-Muster verwendet. Dieses, wie in Gamma u. a. (1996) motiviert, soll die Komplexität des darunter liegenden Systems verbergen. In diesem Fall wird die Abhängigkeit zwischen den Multimedia Diensten und der mobilen Multimedia Middleware mithilfe dieses Musters vermindert.

Um die Modularität der einzelnen Schichten zu erhöhen, wird das *Fassaden*-Muster in jeder der Schichten für die Definition einer einheitlichen Schnittstelle angewendet. Hierdurch wird die interne Komplexität der Schichten nach Außen verborgen, wodurch diese leicht zu verändern sind, ohne dass die anderen Schichten dadurch beeinflusst werden.

Für die Implementierung der Ausführung und Protokollierung der Methode, die ein Multimedia Dienst zur Verfügung stellt, wird ein weiteres Muster verwendet, das so genannte *Befehls*-Muster. Dieses ermöglicht es Anfragen an unbekannte Anwendungen, was bei dem Multimedia Dienst der Fall ist, zu richten und zu protokollieren. Durch die Erweiterung der Befehlsklassenschnittstelle um Laden- und Speichern-Operationen wird ein Logbuch der Vorgänge erstellt. Dieser kann dann bei der Wiederherstellung des Systems durch erneute Ausführung dieser wieder hergestellt werden.

Für das Wiederherstellen des Verbindungszustands wird das *Memento*-Muster verwendet. Durch diesen kann eine Momentaufnahme von dem Multimedia Dienst aufgenommen werden, um später diesen Zustand wieder herstellen zu können. Hiermit wird die Wiederherstellung einer unterbrochenen Verbindung realisiert.

Wie bereits im Entwurf angedeutet, überwacht die Klasse *Connection* die aufgebaute Verbindung, um gegebenenfalls Kommunikationsprobleme aufzuspüren und zu beheben. Für die Realisierung dieser Problemstellung wird das *Beobachter*-Muster, wie es im Java realisiert ist, eingesetzt. Hierzu implementiert die Klasse *Connection* das *Observer*-Interface, wodurch diese über alle Änderungen der beobachteten Verbindung benachrichtigt werden kann.

Bei der Erzeugung der Server-Stellvertreter wird Abstrakt *Fabrik*-Muster verwendet.

### 4.3.2 Umsetzung des Kommunikationsprotokolls

An dieser Stelle wird das im Entwurf bereits genannte Kommunikationsprotokoll realisiert. Auf der Basis des Protokolls wird die Kommunikation zwischen einem Multimedia Dienst und den darauf zugreifenden Client stattfinden. Außerdem werden die für mobile Multimedia

Middleware relevante Infrastrukturinformationen mitübertragen. Dieser, wie bereits erwähnt, baut auf einem XML-Dokument auf um somit die Vorteile, die XML mit sich bring, in die Architektur einfließen zu lassen, wie das bei Web Services zu beobachten ist. Unter Betrachtung der Web Service Technologie, lehnt sich dieser an den dort verwendeten WSDL-Standard an, um die Multimedia Dienste beschreiben zu können. Bei dem Methodenaufruf werden ebenfalls einige Merkmale aus dem SOAP-Standard entnommen.

### Struktur

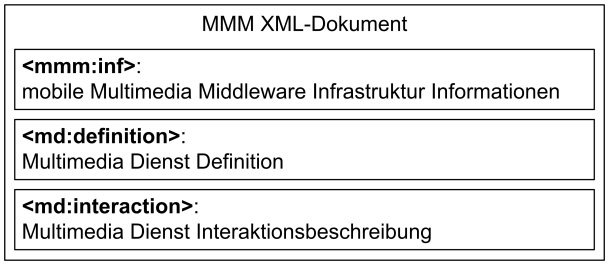


Abbildung 4.3: Grobe Struktur des Kummunikationsprotokolls.

Zuerst sollte die Struktur des Kommunikationsprotokolls festgelegt werden. Wie in der Abbildung 4.3 zu erkennen ist, wird das in der mobilen Multimedia Middleware versendete XML-Dokument in drei Sektionen unterteilt. Jeder dieser Sektionen ist für eine bestimmte Aufgabe zuständig. Bei allen Interaktionen muss die Sektion <*mmm:inf*> gefüllt sein. Die beiden anderen Sektionen werden abhängig von der an den Dienst gestellten Aufgabe gefüllt. So wird bei der Erstanmeldung bei einem Dienst, deren Beschreibung angefordert. Diese wird in die Sektion <*md:definition*> abgelegt. Hat der Client die geforderten Dienstinformationen einmal, kann dieser mit dem Dienst interagieren. Dabei anfallende Daten kommen dann in die Sektion <*md:interaction*> rein. Die interne Struktur der einzelnen Sektionen wird in den folgenden Abschnitten beschrieben.

### Infrasturktur-Protokoll

Bei der Client/Dienst Kommunikation über der mobile Multimedia Middleware ist es notwendig Zustandsinformationen mit zu versenden, die für die Dienstarchitektur relevant sind, sodass hierbei bestimmte Merkmale, wie spontane Verbindung und damit verbundene Mobilität, zustande kommen können. Hierzu wurde die Sektion <*mmm:inf*> zu dem Kommunikationsprotokoll hinzugefügt. Wie man das aus der Abbildung 4.4 entnehmen kann unterteilt sich diese in drei Teile.

In dem Abschnitt <*mmm:env*> kommen physikalische Umgebungsinformationen rein. Die Informationen dienen der Identifizierung der Umgebung und sind bei der Bereitstellung der

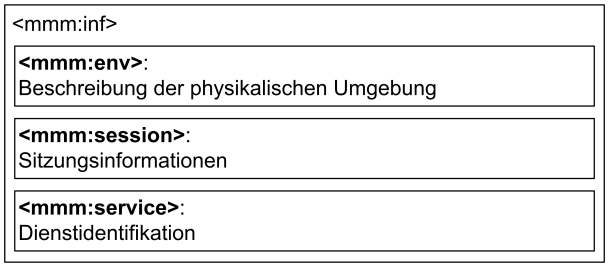


Abbildung 4.4: Sektion: Infrastrukturprotokoll.

Mobilität unabdingbar. Bei einer Kommunikation kann ein Anwender ein oder mehrere Sitzungen mit einem Dienst eröffnen. Um diese eindeutig identifizieren und beschreiben zu können, wird hierfür die Sektion <*mmm:session*> benutzt. Im Weiteren kann ein Benutzer mehrere Dienste gleichzeitig benutzen. Für deren eindeutige Identifikation wurde ebenfalls eine Sektion namens <*mmm:service*> hinzugefügt.

### Dienstdefinitions-Protokoll

In einer dienstorientierten Architektur, so wie diese in (IBM, 2004) beschrieben ist, ist es notwendig, Beschreibungen über die darin befindlichen Dienste zu erstellen und diese nach außen zugänglich zu machen. Durch diese Beschreibung wird ein Zugriff auf die Dienstmethoden ermöglicht. Auch mobile Multimedia Middleware lehnt sich an dieser Vorgehensweise an, um die damit verbundene Flexibilität zu erreichen.

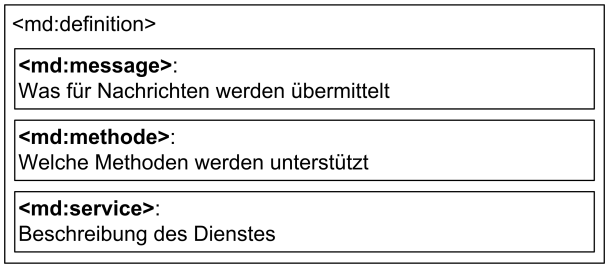


Abbildung 4.5: Sektion: Dienstdefinition.

In der Abbildung 4.5 wird die Struktur des Definitionsabschnittes bildlich dargestellt. Diese lehnt sich an den in Abschnitt 2.3.4 beschriebenen WSDL-Standard und ist in drei Teile unterteilt. Die <*md:message*> Sektion enthält die Nachrichtbeschreibung, welche eine einfache Anfrage- oder Antwortnachricht beschreibt. Diese definiert den Namen der Nachricht und kann aus mehreren Nachrichtelementen bestehen, welche wiederum Nachrichtenparameter oder Rückgabewerte referenzieren können. Die Sektion <*md:methode*> beschreibt die Dienstmethoden, die sich aus mehreren Nachrichtelementen zusammensetzen können. Die Semantik der Methode wird ebenfalls an dieser Stelle angesiedelt sein. Zuletzt kommt die

Sektion <*md:service*>. In dieser werden alle notwendigen Dienstinformationen abgelegt, die den Multimedia Dienst eindeutig semantisch beschreiben.

### Interaktions-Protokoll

Die mobile Multimedia Middleware hat als Hauptaufgabe, eine Interaktion zwischen dem Client und dem Dienst zu ermöglichen. Hierzu wird die Section <*md:interaction*> zur Verfügung gestellt. Diese, wie das in der Abbildung zu sehen ist, setzt sich aus drei Teilen zusammen.

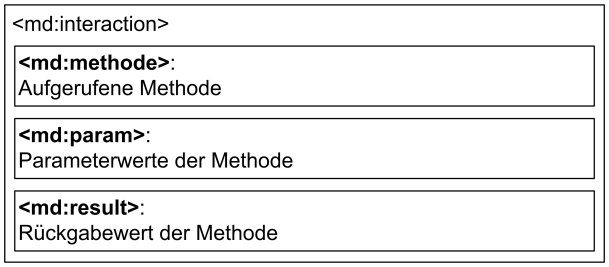


Abbildung 4.6: Sektion: Interaktionsprotokoll.

Ein Multimedia Dienst stellt für einen Anwender verschieden Methoden zur Verfügung, die er ausführen kann. Um eine dieser Methoden aufrufen zu können, muss deren Name spezifiziert werden. Dieses geschieht in der Sektion <*md:methode*>. Es existieren aber auch Methoden, welche für die Ausführung bestimmten Eingabendaten erfordern. Diese werden in der Sektion <*md:param*> spezifiziert. Wurde nun eine Methode erfolgreich ausgeführt bei der ein Rückgabewert zurückgegeben wird, kann dieser in die Sektion <*md:result*> geschrieben werden, sodass der Client diesen empfangen kann.

# Kapitel 5

# Zusammenfassung

„Jeder, der aufhört zu lernen, ist alt, mag er zwanzig oder achtzig Jahre zählen. Jeder, der weiterlernt, ist jung, mag er zwanzig oder achtzig Jahre zählen.“

Henry Ford

Im Abschnitt 5.1 wird eine Zusammenfassung der gesamten Arbeit gegeben. Im Abschnitt 5.2 wird eine Bewertung der Arbeit durchgeführt. Die Bewertung wird anhand der neu gewonnenen Erkenntnisse und mithilfe der hier entwickelten Anwendung gezogen. Dabei werden anschließend Kritikpunkte gesammelt, die während der Bearbeitung der Aufgabenstellung zum Vorschein kamen. Im Abschnitt 5.3 wird ein Ausblick in die Zukunft geben. Dabei werden die Einsatz- und Weiterentwicklungsmöglichkeiten der Software angesprochen.

## 5.1 Zusammenfassung der Arbeit

In dieser Arbeit wurde die Problematik der Mobilität von kleinen mobilen Rechnern betrachtet. Dabei sollte ein System entwickelt werden, welches dabei entstehende Probleme beseitigt und somit die Mobilität der mobilen Rechner ermöglicht. Die hierbei entstehenden Probleme hatten ihren Ursprung in dem ständigen Wechsel der physikalischen Umgebung. Um diese Probleme in Griff zu bekommen, musste das hier entworfene Konzept eine schnurlose Kommunikationstechnologie verwenden. Außerdem wurde ein Sitzungsmanagement entworfen, wodurch die Mobilität ermöglicht wurde.

Hierzu wurde der Leser in der Einleitung in das Umfeld der Problemstellung eingeführt, um deren Natur näher zu verdeutlichen. Hierbei wurden einige fundamentale Sichtweisen, sowie der aktuelle Stand der hierzu notwendigen Technologien, besprochen, wodurch die Tendenz des Rechnens ersichtlich wurde und somit die Notwendigkeit der Bearbeitung der hier gestellten Problemstellung bestätigte. Um dieses noch weiter zu motivieren, wurden des Weiteren einige Zukunftsvisionen und damit verbundene, an diese Entwicklung gestellten, Hoffnungen besprochen. Hierbei hat sich die Aufgabenstellung für diese Arbeit gebildet, die dort ebenfalls eingeleitet wurde.

Von der Aufgabenstellung ausgehend, wurde zu der genaueren Analyse dieser übergegangen. Hierbei sollten die wichtigsten Probleme identifiziert werden, die eine mögliche Realisierung mit sich bringen würde. Dabei wurden drei Einsatzszenarien beschrieben, anhand derer die Probleme ersichtlich wurden. Diese wurden anschließend nach ihrer Relevanz aus der Benutzer- und Informatiksicht kategorisiert. Nachdem die Problemstellung erkannt und genau spezifiziert wurde, wurden die bereits existierenden Konzepte nach möglichen Lösungsansätzen analysiert. In diesem Zusammenhang wurde die Materie der verteilten Systeme und damit verbundenen Middleware ausreichend besprochen, sodass das notwendige Wissen für ein weiteres Verständnis gegeben war. Danach wurden die gängigsten Konzepte oberflächlich beschrieben und einer groben Bewertung, in Hinsicht auf die Einsatzfähigkeit in dem möglichen Lösungsansatz, unterzogen. Hierbei sollten die Konzepte ausgeschlossen werden, die prinzipiell für die hier gestellten Aufgaben nicht geeignet waren. Um eine genauere Bewertung durchführen zu können und somit für die Realisierung die geeigneten Technologien extrahieren zu können, wurden die übrigen Technologien nachfolgend detaillierter beschrieben. Nachdem der Leser mit diesen vertraut gemacht worden ist, wurde deren Bewertung auf die Eignung in einer möglichen Realisierung durchgeführt. In diesem Zusammenhang wurden hierzu notwendige Bewertungskriterien ermittelt. Darauf basierend wurden die in Betracht gezogenen Technologien evaluiert. Bei der anschließenden Diskussion der Bewertungsergebnisse wurden die für die Realisierung des Systems geeigneten Techniken ausgewählt.

Mit den Erkenntnissen der Analyse wurde dann der Entwurf des Systems durchgeführt. Dabei wurden als Erstes die Randbedingungen, die an das System gestellt wurden, aus Software- sowie Hardwaresicht festgelegt. Diese ergaben sich aus den Einsatzbedingungen, welche wiederum ihren Ursprung in den Produktanforderungen fanden. Als Nächstes wurden konzeptionelle Entscheidungen beschrieben. Dabei wurden die Ansprüche, die die mobile Multimedia Middleware an die Datenhaltung, Middleware, Benutzerschnittstellen sowie deren Verteilung in dem Netzwerk gestellt hat, betrachtet. Auf den gestellten Anforderungen aufbauend, wurde zu dem Architekturentwurf übergegangen. Dabei wurden die in der Analyse erkannten Anwendungsfälle weiter verfeinert und die daraus identifizierten Systemkomponenten abgeleitet. Durch die weitere Verfeinerung wurde eine grobe Architektur des Systems ermittelt. Daraus hervor gehende Schichtenarchitektur, die darin enthaltenen Schichten und die Beziehungen zwischen diesen wurden als Nächstes besprochen. Nachdem das System in seinen Einzelheiten entworfen wurde, wurde die Dienstkommunikation diskutiert. Zum Schluss wurden einige Testszenarien beschrieben, durch die die Qualität des Systems sichergestellt werden kann.

Ausgehend von dem Entwurf und den dort gewonnenen Erkenntnissen wurde dann zu der Realisierung des Systems übergegangen. Dabei wurde zuerst die Testumgebung in ihren Einzelheiten beschrieben. An dieser Stelle wurden die verwendeten Software- und Hardwarekomponenten sowie deren Auswahl erläutert. Danach wurde die Umsetzung des entworfenen Systems beschrieben. Hierbei wurde die Verwendung der Entwurfsmuster in den

einzelnen Systemkomponenten motiviert. Anschließen wurde das im Entwurf besprochene Dienstkommunikationsprotokoll in seinen Einzelteilen umgesetzt.

## 5.2 Bewertung

In diesem Abschnitt sollten die Erkenntnisse und der Stand der Entwicklung festgehalten werden. Dabei wird anschließend eine Bilanz über die in dieser Arbeit gesammelten Erfahrungen gezogen.

### 5.2.1 Stand der Entwicklung

Es wird als Erstes festgehalten, dass eine Middleware für mobile Clients erfolgreich umgesetzt wurde, sodass das zum Anfang vorgestellte Szenario mithilfe der hier entworfenen Architektur ermöglicht werden kann. Hierdurch wurde auf einer höheren Abstraktionsebene angesiedelte Kommunikation ermöglicht als die, die von CORBA bereitgestellt wird. Eine Middleware für dienstorientiertes, allgegenwärtiges Rechnen wurde realisiert.

Das hierbei entworfene System entstand unter der Berücksichtigung der Konstruktionsregeln des Software-Engineering. Das System zeichnet sich durch den modularen Aufbau und klar definierter Schnittstellen robust und flexibel. Die Problematik der Sicherheit wurde nur theoretisch betrachtet, sodass an dieser Stelle noch weiter gearbeitet werden muss. Der automatisierte Verbindungsaufbau wurde theoretisch entworfen, konnte aber wegen fehlender Schnittstellen zu dem Funkmodul nicht nachgebildet werden. Des Weiteren wurden prototypische Dienste und ein Client entwickelt, wodurch das System zum größten Teil getestet werden konnte. So lässt sich sagen, dass die am Anfang gestellten Anforderungen zum größten Teil erfüllt wurden.

### 5.2.2 Gewonnene Erkenntnisse

Es wurde durch die Arbeit gezeigt, dass die hier gestellte Problemstellung bereits heute realisierbar ist. Durch die hier entwickelte Middleware wurde eine Plattform geschaffen, auf der das spontane Vernetzen und somit die Mobilität ermöglicht wurde. Dabei wurden neu entstehende Probleme identifiziert, die durch die Vorgaben hervorgerufen wurden.

So zeigte diese Arbeit, dass das Hauptproblem in dem Wechsel der einzelnen Zugangspunkte liegt. Die hierbei entstehende physikalische Verbindungsunterbrechung durfte keine Auswirkungen auf die logische Verbindung haben. Dabei wurde erkannt, dass dieses durch ein Sitzungsmanagement zu erreichen ist, wodurch eine logische Verbindung weiter bestehen konnte, obwohl gerade ein Zugangspunktwechsel stattfand.

Es hat sich ebenfalls gezeigt, dass bereits Technologien existieren, die eine mobile Architektur unterstützen. Dabei haben sich CORBA und JAVA in Verbindung mit XML und

der Bluetooth-Technologie als solche erwiesen. Diese im Zusammenspiel miteinander erfüllten die gestellten Anforderungen, sodass eine allgegenwärtige, dienstorientierte Middleware möglich wurde.

Außerdem stellte sich heraus, dass die Verwendung von XML als Transportmedium für Dienstanfragen und Middleware-Infrastrukturinformationen dieses Konzept flexibel für Erweiterungen machte. Dieses wurde noch weiter durch die intern verwendete schichtenorientierte Architektur bekräftigt. Dabei wurden die Schichten auf beiden Kommunikationsseiten symmetrisch angeordnet. Hierbei wurde ein modularer Systemaufbau erreicht, wodurch der Austausch der einzelnen Schichten vereinfacht wurde, da der Austausch einer Schicht die andere nicht beeinträchtigt.

Zusätzlich hierzu wurde durch den Einsatz genannten Technologien die geforderte Offenheit und Portabilität gewährleistet, wodurch die beiden Kommunikationsseiten voneinander weitgehend entkoppelt werden konnten. Java rundete diese Architektur ab, indem es sie plattformunabhängig machte. Hierdurch ist diese überall einsetzbar, wo die JVM unterstützt wird.

Durch die interne Ortstransparenz, die durch CORBA gegeben wird, wird eine Skalierbarkeit der Architektur möglich. So können die Dienste auf mehrere Rechner hinter dem Zugangspunkt verteilt werden. Außerdem wird die Entwicklungszeit der Clients hierdurch verkürzt, da dieser nicht die Suche nach den geeigneten Diensten implementieren muss. Das macht den Client und somit die Interaktion mit den Diensten stabiler.

Wie bereits besprochen erlaubt die hier entworfene Architektur eine höher angesiedelte Kommunikation, wodurch eine dienstorientierte Plattform geschaffen wurde. Auf dieser kommunizieren Dienste und keine einzelnen Objekte mehr miteinander.

### 5.2.3 Kritik

Eines der Hauptprobleme war die Bereitstellung der automatisierten Verbindung mit den Bluetoothgeräten. Hierzu war eine Bluetooth-Schnittstellenimplementierung notwendig. Zum Zeitpunkt der Bearbeitung dieser Arbeit war aber nur eine kommerzielle Lösung verfügbar. Aus Kostengründen konnte diese im Rahmen dieser Arbeit nicht eingesetzt werden und wurde somit nicht berücksichtigt. Daher musste die Verbindung manuell durchgeführt werden. Hierdurch ist das Szenario mit der sauberen Mobilität nicht richtig gegeben.

Die Performance der Anwendung war ebenfalls nicht akzeptabel. Diese spielte aber in Rahmen dieser Arbeit keine Rolle, da man davon ausgehen kann, dass die Geräte in kurzer Zeit die hierzu notwendige Leistung bereitstellen werden.

Die am Anfang geforderte Unwissenheit über den Ausführungskontext wurde teilweise erfüllt. So ist intern durch die Minimierung der Client/Dienst-Schnittstelle dieses gegeben. Wenn aber eine Darstellungstechnik für die Dienste entwickelt wird, stößt man auf Probleme. Um Dienste entsprechend der Gegebenheiten darstellen zu können, müssen die beiden Kommunikationspartner gegenseitig Wissen voneinander besitzen, um sich auf die Beson-

derheiten anpassen zu können.

## 5.3 Ausblick

Eine Kommunikationsmiddleware ist dann vollendet, wenn es eine Übertragungssicherheit gewährleistet. Aus diesem Grund sollte die Sicherheit der Übertragung zwischen dem mobilen Client und dem Multimedia Dienst erarbeitet werden. Wie schon in Kapitel 2 erwähnt, stellt eine so offene Architektur auch viele neue Türen für Unbefugte offen. Diese müssen identifiziert und abgeschottet werden, damit dieses System eine Plattform für Dienste wird, bei dem kritische Daten sicher übertragen werden können. Einige Möglichkeiten wurden in der Diplomarbeit (Lüpke, 2004) von André Lüpke vorgestellt.

Die Beschreibung der Multimedia Dienste stellt ebenfalls eine Herausforderung dar. So muss eine Beschreibungstechnik erarbeitet werden, die die Dienste vollständig beschreibt, auf der anderen Seite, eine einheitliche Verarbeitung und Suche ermöglicht. Eine Alternative stellt WSDL dar. Diese und vergleichbare Ansätze sollten weiter nach geeigneten Lösungsverfahren abgesucht werden, die bei der Beschreibung der Dienste helfen könnten.

Im Zusammenhang mit dem Aufbau der Verbindung sollte weiter die bei der Mobilität unterschiedlich auftretende Bewegungsgeschwindigkeit überprüft werden. Hierbei muss überprüft werden, wie schnell das mobile Gerät sich von einem Zugangspunkt zu anderen bewegen kann. Möglicherweise wird dieses bereits durch die Bluetooth-Technologie bestimmt.

Wie das in der Web Service Technologie der Fall ist, so tritt auch hier das Problem der Klassifizierung der Dienste auf. Hier sollte ein Ansatz erarbeitet werden, welcher diese für den Benutzer kanonisiert, sodass dieser mit der Vielfältigkeit der Dienste umgehen kann.

Ein weiteres Problem stellt die einheitliche Darstellung von unterschiedlichen Diensten dar. In der dienstorientierten Umgebung kommen unterschiedlichste Dienste mit unterschiedlichsten Dienstleistungen vor. Der Benutzer sollte im Stande sein, diese durch kurze Einarbeitungszeit benutzen zu können. Ist dieses nicht gegeben, hat die allgegenwärtige, dienstorientierte Architektur keine Chance sich durchzusetzen. So muss eine Technik entwickelt werden, mit der dieses Problem behoben wird.

Die physikalische Verbindung, wie schon im Kapitel 2 erkannt, wird sehr wahrscheinlich in einer unterschiedlichen Qualität vorliegen. Damit ist z.B. die Stabilität der Verbindung gemeint. Im diesem Sinne sollte vielleicht ein Mechanismus entwickelt werden, mit dessen Hilfe man diese Eigenschaft auswertet und gegebenenfalls hierfür notwendigen Vorkehrungen trifft.

Wie im Kapitel 3 bereits erwähnt, wurde hier nur ein Prototyp des Systems entwickelt. Dieser realisiert ein Ein-Benutzersystem. Im realen Leben ist dieses aber nicht akzeptabel. Hierbei sollte das Konzept um notwendige Mechanismen erweitert werden, die eine gleichzeitige Benutzung von mehreren Benutzern erlaubt.

Es muss weiterhin analysiert werden, wie weit Multimedia Dienste existieren werden, die

direkt auf dem mobilen Client ausgeführt werden sollen. Dabei soll untersucht werden, welche Rolle die mobilen Multimedia Middleware in diesem Zusammenhang spielt.

Eine weitere Aufgabe, die noch ansteht, ist die Verfeinerung des Protokolls und die darin enthaltenen Informationen. Dieser muss genauer analysiert und festgelegt werden.

# Literaturverzeichnis


[Babic 2003] BABIC, Alexander: *Diplomarbeit:Entwicklung einer profilverarbeitenden übiquitären Anwendung*. Hochschule für angewandte wissenschaften, Fachbereich Elektrotechnik und Informatik, Hamburg, 2003

[Balzert 2000] BALZERT, Helmut: *Lehrbuch der Software-Technik. Software-Entwicklung*. 2. Auflage. Heidelberg, Berlin : Spektrum Akademischer Verlag, 2000. – ISBN 3-8274-0480-0

[Bengel 2002] BENGEL, Günther: *Grundkurs Verteilte Systeme: Grundlagen und Praxis des Client-Server-Computing*. 2. Auflage. Wiesbaden : Vieweg, 2002. – ISBN 3-528-25738-5

[Chappell und Jewell 2003] CHAPPELL, David A. ; JEWELL, Tyler: *Java Web Services*. 1. Auflage. Beijing u.a. : O'Reilly, 2003. – ISBN 3-89721-284-6

[Coulouris u. a. 2002] COULOURIS, George ; DOLLIMORE, Jean ; KINDBERG, Tim: *Verteilte Systeme: Konzepte und Design*. 3. überarbeitete Auflage. München : Pearson Studium, 2002. – ISBN 3-8273-7022-1

[Dawson 2003] DAWSON, Christian W.: *Computerprojekte im Klartext*. München : Pearson Studium, 2003. – ISBN 3-8273-7067-1

[Doss 2000] DOSS, George M.: *CORBA developer's guide with XML*. Plano, Tex. : Wordware Publ., 2000. – ISBN 1-556-22668-3

[Eberhart und Fischer 2001] EBERHART, Andreas ; FISCHER, Stefan: *Java-Bausteine für E-Commerce-Anwendungen: verteilte Anwendungen mit Servlets, EJB, CORBA und XML und SOAP*. 2. aktualisierte und erw. Aufl. München : Hanser, 2001. – ISBN 3-446-21732-0

[Eclipse 2004] ECLIPSE FOUNDATION (Hrsg.): *Eclipse*. 2004. – URL `http://www.eclipse.org`. – Zugriffsdatum: 2004-04-24

[Gamma u. a. 1996] GAMMA, E. ; HELM, R. ; JOHNSON, R. ; VLISSIDES, J.: *Entwurfsmuster: Elemente wiederverwendbarer objektorientierter Software*. Bonn : Addison-Wesley, 1996. – ISBN 3-89319-950-0

[Gruhn und Thiel 2000] GRUHN, Volker ; THIEL, Andreas: *Komponentenmodelle: DCOM, JavaBeaans, EnterpriseJavaBeans, CORBA*. 2.Auflage. München : Addison-Wesley, 2000. – ISBN 3-8273-1724-X

[Hansmann u. a. 2001] HANSMANN, Uwe ; NICKLOUS, Scott ; MERK, Lothar ; STOBER, Thomas: *Pervasive Computing Handbook*. 1. Auflage. Heidelberg : Springer Verlag, 2001. – ISBN 3-540-67122-6

[Heinrich 2003] HEINRICH, Hayat: *Diplomarbeit:Auswahl einer Middleware für mobile Clients*. Hochschule für angewandte wissenschaften, Fachbereich Elektrotechnik und Informatik, Hamburg, 2003

[HP 2004] HEWLETT-PACKARD DEVELOPMENT COMPANY, L.P. (Hrsg.): *HP*. 2004. – URL `http://www.hp.com`. – Zugriffsdatum: 2004-04-24

[IBM 2004] IBM CORPORATION (Hrsg.): *The evolution of Web applications into service-oriented components with Web services*. 2004. – URL `http://www-106.ibm.com/developerworks/webservices/library/ws-tao/`. – Zugriffsdatum: 2004-05-08

[Insignia 2004] INSIGNIA SOLUTIONS INC. (Hrsg.): *JEODE Java Runtime Environmen*. 2004. – URL `http://www.insignia.com/content/about/releases/011126.shtml`. – Zugriffsdatum: 2004-04-24

[IZT u. a. 2001] IZT ; SFZ ; IAT: Entwicklung und zukünftige Bedeutung mobiler Multimediadienste / IZT-Institut für Zukunftsstudien und Technologiebewertung, SFZ-Sekretariat für Zukunftsforschung, IAT-Institut Arbeit und Technik. URL `http://www.izt.de/pdfs/IZT_WB49_Mobile_Multimediadienste.pdf`, 2001 (WerkstattBericht Nr. 49). – Forschungsbericht

[JacORB 2004] FREIE UNIVERSITÄT BERLIN AND XTRADYNE TECHNOLOGIES AG (Hrsg.): *JacORB*. 2004. – URL `http://www.jacorb.org`. – Zugriffsdatum: 2004-04-24

[James F. Kurose 2002] JAMES F. KUROSE, Keith W. R.: *Computernetze: Ein Top-Down-Ansatz mit Schwerpunkt Internet*. Pearson Studium, 2002. – ISBN 3-8373-7017-5

[JAXP 2004] SUN MICROSYSTEMS, INC. (Hrsg.): *Java API fo XML Processing*. 2004. – URL `http://java.sun.com/xml/jaxp/index.jsp`. – Zugriffsdatum: 2004-04-24

[Lüpke 2004] LÜPKE, André: *Diplomarbeit:Entwurf einer Sicherheitsarchitektur für den Einsatz mobiler Endgeräte*. Hochschule für angewandte wissenschaften, Fachbereich Elektrotechnik und Informatik, Hamburg, 2004

[Mark Weiser 2004] WEISER, Mark: *Ubiquitous Computing*. 2004. – URL `http://www.ubiq.com/hypertext/weiser/UbiHome.html`. – Zugriffsdatum: 2004-05-08

[Mörike 2003] MÖRIKE, Michael (Hrsg.): *Entwicklungsplattformen - Java versus .NET*. Heidelberg : dpunkt-Verl., 2003. – ISBN 3-89864-202-X

[OMG 2004a] : *CORBA*. 2004. – URL `http://www.omg.org/technology/documents/spec_catalog.htm`. – Zugriffsdatum: 2004-03-24

[OMG 2004b] : *OMA*. 2004. – URL `http://www.omg.org/oma/`. – Zugriffsdatum: 2004-04-28

[Orfali und Harkey 1998] ORFALI, Robert ; HARKEY, Dan: *Client/Server Programming with Java and Corba*. Second Edition. Wiley Computer Publishing, 1998

[Parnas 1972] PARNAS, David L.: On the Criteria To Be Used in Decomposing Systems into Modules. In: *Communications of the ACM* 15 (1972), Dezember, Nr. 12, S. 1053–1058. – ISSN 0001-0782

[PersonalJava 2004] : *PersonalJava*. 2004. – URL `http://java.sun.com/products/personaljava/`. – Zugriffsdatum: 2004-04-24

[Revout 2003] REVOUT, Ilia: *Diplomarbeit:Design und Realisierung eines Frameworks für Bildverarbeitung*. Hochschule für angewandte wissenschaften, Fachbereich Elektrotechnik und Informatik, Hamburg, 2003

[Schlichting 2003] SCHLICHTING, Marco: *Diplomarbeit: Intergration einer Bluetooth Schnittstelle in eine bestehende Embedded Applikation.* Hochschule für angewandte wissenschaften, Fachbereich Elektrotechnik und Informatik, Hamburg, 2003

[Schneider und Werner 2001] SCHNEIDER, Uwe ; WERNER, Dieter: *Taschenbuch der Informatik*. 4.Auflage. Fachbuchverlag Leipzig, 2001

[SOA 2004] : *SOA*. 2004. – URL `http://www.serviceoriented.org/service_oriented_architecture.html`. – Zugriffsdatum: 2004-05-08

[Steinweg 2002] STEINWEG, Carl: *Projektkompass Softwareentwicklung - Geschäftsorientierte Entwicklung von IT-Systemen*. 4,überarbeite und erweiterte Auflage. VIEWEG, März 2002

[SUN 2004a] : *Data Binding*. 2004. – URL `http://java.sun.com/xml/jaxp/dist/1.0.1/docs/binding/DataBinding.html`. – Zugriffsdatum: 2004-05-15

[SUN 2004b] : *JINI*. 2004. – URL http://www.sun.com/software/jini/overview/index.html. – Zugriffsdatum: 2004-03-24

[Tanenbaum und van Steen 2002] TANENBAUM, Andrew S. ; STEEN, Maarten van: *Distributed Systems: Principles and Paradigms*. Upper Saddle River, NJ : Prentice Hall, 2002. – ISBN 0130888931

[TDKSystems 2004] : *TDKSystems*. 2004. – URL (http://www.tdksystems.com). – Zugriffsdatum: 2004-04-24

[UserLand Software, Inc. 2004] : *XML - RPC*. 2004. – URL http://www.xmlrpc.com/. – Zugriffsdatum: 2004-03-24

[Vonhoegen 2004] VONHOEGEN, Helmut: *Einstieg in XML*. Galileo Computing, 2004. – ISBN 3-89842-488-X

[W3C 2004a] : *SOAP Version 1.2*. 2004. – URL http://www.w3.org/TR/2003/REC-soap12-part0-20030624/. – Zugriffsdatum: 2004-03-24

[W3C 2004b] W3C: *Web Services: Architektur*. online. 2004. – URL http://www.w3.org/TR/2003/WD-ws-arch-20030514/

[W3C 2004c] W3C: *Web Services: Glossar*. online. 2004. – URL http://www.w3.org/TR/2003/WD-ws-gloss-20030514/

[Weiser 1991a] WEISER, Mark: The computer for the 21st century. In: *Scientific American* 265 (1991), sep, Nr. 3, S. 94–104. – URL http://www.ubiq.com/hypertext/weiser/SciAmDraft3.html. – Zugriffsdatum: 2004-05-08

[Weiser 1991b] WEISER, Mark: *Some computer science issues in ubiquitous computing*. Bd. 36 Nr.7. Comms ACM, 1991. – 74–84 S

[Weiser 1993] WEISER, Mark: Some Computer Science Issues in Ubiquitous Computing. In: *Scientific American* (1993), march. – URL http://www.ubiq.com/hypertext/weiser/UbiCACM.html. – Zugriffsdatum: 2004-05-08

[Westphal 2001] WESTPHAL, Ralf: *.NET kompakt*. Spektrum Akademischer Verlag, 2001

# Versicherung über Selbstständigkeit

Hiermit versichere ich, dass ich die vorliegende Arbeit im Sinne der Prüfungsordnung *Softwaretechnik PO 98* nach §24(5) ohne fremde Hilfe selbstständig verfaßt und nur die angegebenen Hilfsmittel benutzt habe.

Ort, Datum

Unterschrift des Studenten