Matthias Pisarkiewicz

Entwicklung eines mobilen location based Webservice (Apache-Axis-Server J2ME/KSOAP-Client) und einer Webservice Registry auf Basis von MapPoint

Diplomarbeit eingereicht im Rahmen der Diplomprüfung
Im Studiengang Softwaretechnik
Am Studiendepartment Informatik
Der Fakultät Technik und Informatik
der Hochschule für Angewandte Wissenschaften Hamburg

Betreuender Prüfer : Dipl. Inform. Birgit Wendholt
Zweitgutachter : Prof. Dr. rer. nat. Kai von Luck

Abgegeben am 30. August 2006

Matthias Pisarkiewicz

Thema der Diplomarbeit
Entwicklung eines mobilen location based Webservice (Apache-Axis-Server J2ME/kSOAP-Client) und einer Webservice Registry auf Basis von MapPoint

Stichworte
Webservices, mobile Webservices, kSOAP, ortsbasierte Webservices, mobiler smart Client, Java, MIDP, ortsbezogener Verzeichnisdienst, Registry für Webservices, MapPoint Server, ortsbasierte Dienstleistungssuche, Umgebungssuche, kartenbasierter Routenplaner, geographische Informationssysteme, location based services

Kurzzusammenfassung
Gegenstand dieser Diplomarbeit ist die Durchführung einer Machbarkeitsstudie und der daraus abgeleitete Entwurf eines ortsbasierten Webservice für einen mobilen Client auf Basis vom Java-MIDP kSOAP. Insbesondere soll ein ortsbezogener Verzeichnisdienst für Webservices konzipiert werden. Zur Realisierung der ortsbezogenen Eigenschaften, werden bestehende geographische Informationssysteme, andere relevante Werkzeuge und Entwicklungsumgebungen hinsichtlich ihrer Eignung für den Aufbau eines derartigen Systems besprochen und bewertet. Es wird gezeigt, dass MapPoint eine mit Einschränkungen geeignete Infrastruktur für die Implementierung einer ortsbasierten Registry für Webservices ist. Als Anwendungsbeispiel wurde die Buchung von Booten in der näheren Umgebung des Standorts des Nutzers gewählt. Dazu gehört die Suche, Verfügbarkeitsprüfung und ein kartenbasierter Routenplaner. Der Entwurf motiviert aus Gründen der Ressourcen-beschränkung einen smart Client mit einer serverseitigen Proxyarchitektur. Die Machbarkeit des gewählten Ansatzes wird in einer durchgängigen Implementierung vom mobilen Client über den Proxy bis hin zu den Businessservern, darunter auch der MapPoint Server, nachgewiesen. Dabei werden die Beschränkungen der verwendeten Software, insbesondere die von MapPoint und kSOAP, für die Lösung der Aufgabenstellung herausgearbeitet.

Matthias Pisarkiewicz

Title of the paper
Development of a mobile location based webservice (Apache-Axis-Server J2ME/kSOAP-Client) and a webservice registry, based on MapPoint.

Keywords
webservices, mobile webservices, kSOAP, location based webservices, mobile smart client, java, MIDP, J2ME, location based registry, registry for webservices, MapPoint server, location based service finding, find nearby, map based route planer, geographical information systems, location based services

Abstract
The subject of this assignment is a feasibility study of the location based webservice for the mobile client based on Java-MIDP kSOAP. The main concept investigated is the location based service registry. Geographic information systems will be examined according to their suitability for the project, as well as other tools and development environments. We will show that MapPoint is a suitable infrastructure for the implementation of location based registry for webservices. We will show an example of an application of this concept for a boat reservation system for customers being in the area of the boat rentals. The example covers boat searching feature, verification of boat availability, and map-based routing information. Becasue of the

limited resources, the architecture of the system will integrate a proxy on the server side. We will prove that the concept presented in this project can be implemented starting with the mobile client, through a proxy, to business servers, including a MapPoint server. We will also cover the limitations of the software used, especially Map Point and kSOAP, and how it affects our project.

Danksagung

Ich danke hiermit allen, die mich während der Diplomarbeit unterstützt haben.

Ich danke meinem Herrn Jesus Christus, für die Kraft und Ausdauer, die Er mir zur Lösung der Aufgaben gegeben hat.

Prof. Birgit Wendholt, danke ich für die Anregungen und Kritik.

Ein großes Dankeschön geht an meine liebe Frau Magdalena und meine kleinen Kids, David und Livia. Sie mussten mich besonders oft entbehren.
Auch meinen Eltern bin ich sehr dankbar, da sie mich, besonders in der Endphase des Projektes, stark motiviert haben.

Bei meinem guten Freund, Piotr Wendt, möchte ich mich für die Korrektur einiger Kapitel der Arbeit bedanken.

Inhaltsverzeichnis

1. Einleitung

Motivation

In Zeiten der ubiquitären[1] Computertechnik, in denen wir heute leben, sind mobile Geräte, wie **PDAs**, **Handys** oder **Laptops** zu unseren ständigen Begleitern geworden.

Ein System soll entwickelt werden, welches bestimmte Standorte oder Bereiche auf der Erde lokalisieren und graphisch präsentieren kann. Objekte mit mehreren Attributen sollen abgelegt werden können, mit der Möglichkeit auf diese zugreifen und sie graphisch darstellen zu können. Für den Zugriff sollen mobile Geräte verwendet werden. Mobile Geräte erfreuen sich hoher Beliebtheit, vor allem wegen ihrer kleinen Größe und hoher Mobilität d.h. Ortsungebundenheit. Der Benutzer kann deshalb von fast überall telefonieren bzw. im Internet surfen und verschiedene Dienste nutzen. Die wohl wichtigste Eigenschaft mobiler Geräte ist die erwähnte hohe Mobilität. Aber auch der direkte Zugriff auf persönliche Daten bietet einen Mehrwert an Komfort. Außerdem fällt dem Nutzer der Umgang mit einem persönlichen Gerät leichter und schneller als mit einem fremden Gerät, das beispielsweise in einem Büro oder einem Internetkaffe steht. Ein Jungunternehmer beispielsweise, möchte mobil sein und seine Geschäfte nicht nur vom Büro aus abwickeln können. Die Mobilität soll zumindest dadurch gegeben werden, dass eine Verbindung ins Internet und damit auch zu unserem System von Zuhause oder vom Büro aus, bzw. wo auch immer ein internetfähiger Rechner steht, ermöglicht wird.

Bei der großen Anzahl an Anwendungen für mobile Geräte, muss die zu entwickelnde Anwendung etwas Besonderes bieten. Das Besondere an der zu entwickelnden Anwendung soll die Art der Anwendung sein, und zwar soll diese in die Klasse der ortsbasierten Services (**location-based services**) fallen. Mit location-based-services ist man in der Lage, dem Nutzer, passend für seinen Standort, optimale Dienste anzubieten. Mit wachsender Entwicklung mobiler Kommunikation sind ortsbasierte Systeme immer mehr gefragt. Die Nutzer derartiger Systeme können Umgebungssuchen starten, Routen ermitteln und die Ergebnisse in Kartenform anzeigen lassen. Diese Eigenschaften mobiler Anwendungen machen das mobile Gerät zu einem attraktiven Begleiter.

Ortsbasierte Services sind Dienste, die eine Basis von ortsbezogenen Daten verwenden, weshalb sie auf so genannten **geographischen Systemen (GIS)** aufbauen. Dieser Diplomarbeit kam zu gute, dass vor Kurzem ein ortsbasierter Service mit eigener Datenbasis auf dem Markt erschienen ist. Dies ist der **MapPoint-Webservice** der Firma Microsoft. Dieser soll in der Arbeit genauer untersucht und im praktischen Teil verwendet werden.

Bisher haben wir von Services bzw. von Diensten geredet. Heute gibt es die Möglichkeit Dienstleistungen als so genannte **Webservices** technisch umzusetzen. Bereits die Art, wie Webservices verwaltet werden, spricht dafür diese einzusetzen. Sie werden nämlich als fertige Komponenten genutzt. Unabhängige Dienstleister bieten dabei die Webservices an und müssen diese nicht lokal lagern. Dafür können die Rechner eines Application-Service-Providers (ASP) genutzt werden. Des Weiteren zeichnen sich Webservices durch einheitliche und verbreitete Protokolle und Schnittstellen aus. Es wird zum Großen Teil HTTP und XML verwendet. Der Grundgedanke bei der Verwendung von Webservices ist der, die Anwendungsfunktionalität, die im Internet vorhanden ist, auf eine einfache Art und Weise verfügbar zu machen. Sie ermöglichen es, bereits bewährte Softwarekomponenten Webservice fähig zu machen und in das Webservice-Modell zu integrieren. Diese Eigenschaft dieser Technologie macht sie sehr attraktiv, da hierdurch im hohen Maße die Wiederverwendung unterstützt wird.

[1] von Mark Weisner 1992 eingeführter Begriff der allgegenwärtigen, verschwindenden Computer – engl. ubiquitous computing bzw. ambient computing

An dieser Stelle könnte man fragen, wie Webservices, die irgendwo im weiten Internet verstreut liegen, aufzufinden sind. Am einfachsten wäre es, wenn sich die Beteiligten (also der bzw. die Serviceanbieter und die Servicenutzer) bereits kennen gelernt und ausgetauscht hätten. Diese Art der Bekanntmachung von Services ist jedoch keine geeignete Grundlage für den Einsatz von Webservices als eine globale Dienstplattform. Es sollte nämlich möglich sein, Anwendungen aus einer Vielfalt weltweit verfügbarer für den Nutzer zuerst unbekannter Dienste aufzubauen. Dafür gibt es so genannte **Verzeichnisdienste**, die so genannten **Registries**, welche die Registrierung von Webservices und die Suche nach diesen anhand von beschreibenden Merkmalen ermöglichen. Das zu entwickelnde System soll eine **ortsbasierte Registry** anbieten.

Als eine weitere Motivation für den Sinn und Zweck dieser Diplomarbeit wird ein Szenario einer möglichen Anwendung vorgestellt, bei der von einem mobilen Gerät auf mehrere Webservices zugegriffen wird. Dieses Szenario wird für den praktischen Teil der Diplomarbeit, als Grundlage, verwendet.

Beispiel einer Anwendung: mobiler ortsbasierter Bootsverleih
Ein viel beschäftigter Jungunternehmer aus Hamburg plant fürs kommende Wochenende einen Betriebsausflug. Für die nächste Woche steht ein besonders wichtiges, kapitalstarkes und besonders schwieriges Projekt an. Der Ausflug soll dazu dienen, in einer entspannten Atmosphäre den Ablauf des Projektes zu besprechen und die Mitarbeiter zu motivieren. Eine Bootsfahrt auf der Elbe, bei schönem Wetter, wäre wohl besonders geeignet. Da die Firma, noch nicht besonders lange existiert und in letzter Zeit besonders viel investiert wurde, gab es bisher keine Möglichkeit sich ein eigenes Boot zuzulegen, deshalb wird der Unternehmer auf das vielfältige Angebot der Hamburger Bootsverleihe, wie z.B. Yachtclubs ausweichen müssen. Er beauftragt den mobilen Wochenplaner, fürs kommende Wochenende den Ausflug zu planen, vorausgesetzt das Wetter spielt mit, was bedeutet, dass ein Wetter-Webservice konsultiert werden muss. Sobald ein geeignetes Boot gefunden und gebucht wurde, werden alle Mitarbeiter per Email benachrichtigt. Einige Vorgaben, wie die Bootsart, Motorkraft, die Suchregion und die maximale Teilnehmerzahl, müssen der Buchungssoftware übergeben werden.
Der mobile Wochenplaner beginnt mit der Suche nach verfügbaren Booten in der vorgegebenen Umgebung und verwendet einen Routenplaner für die Wegbeschreibung zum entsprechenden Bootsverleih. Anschließend stellt er die Route nach ortsbezogenen Standortinformationen graphisch dar.
Aufbauend auf diesem Beispiel, wird in der Arbeit ein System für ortsbezogene Webservices entworfen und entwickelt.

Gliederung
Die im Rahmen der Diplomarbeit zu entwickelnde Applikation bzw. der Prototyp soll mit Hilfe der bekannten Software-Engineering-Methoden entwickelt werden. Vor der eigentlichen Implementierung muss Marktrecherche betrieben werden um die Arbeit richtig einordnen zu können. Durch die Marktrecherche wird es möglich sein, Aussagen über die Wahl der Entwicklungsumgebung, d.h. der Werkzeuge zur Softwareentwicklung aber auch Softwarekomponenten, die man bereits nutzen kann, zu treffen. Für die technische Realisierung soll eine besondere Art der Verteilung genutzt werden, nämlich die Webservicetechnologie, die im Kapitel 2.2 vorgestellt wird. In Kapitel 2.2 wird auch ein Vergleich von Webservices mit den bisher bekannten Verteilten Systemen unternommen und die Vorteile der Webservicetechnologie hervorgehoben. Das Besondere an dem zu entwickelnden System soll der Einsatz eines speziellen Verzeichnisdienstes sein, der, wie bereits in der Einleitung erwähnt wurde, die Anforderung erfüllen muss, die Ortsabhängigkeit

zu unterstützen. In Kapitel 2.1 werden vergleichbare Arbeiten analysiert. Unter anderem, geographische Informationssysteme und der MapPoint-Server. Dieser soll daraufhin untersucht werden, ob er allen unseren Anforderungen gerecht wird. Es wird untersucht, ob bestehende Verzeichnisdienste, wie UDDI, über ortsbezogene Eigenschaften verfügen bzw., ob bestehende GIS-Systeme die Eigenschaften eines Verzeichnisdienstes erfüllen oder um solche erweitert werden können. Der MapPoint-Server besitzt eine Webserviceschnittstelle, was besonders gut in die geplante serviceorientierte Umgebung dieser Arbeit passen wird. Weitere Eigenschaften von GIS und MapPoint werden im Vergleichenden Kapitel (Kap.2) aufgeführt und verglichen. Kapitel 3 untersucht Laufzeitumgebungen hinsichtlich des Einsatzes von Webservices auf mobilen Geräten. Im Abschnitt 4-Analyse wird die Software geplant, die Anforderungen (Kap. 4.1) werden analysiert, woraus Anwendungsfälle mit geeigneten Szenarien entstehen sollen. Diese findet man im Kapitel 4.2. Entsprechende Komponenten sind im Kapitel 4.3 beschrieben. Die gesamte Architektur wird im Kapitel 4.4 vorgestellt, unter anderem wird auch der Einsatz Java fähiger mobiler Geräte berücksichtigt. Nach der Analyse müssen Details ausgearbeitet werden. Technische Komponenten werden eingeführt und erläutert. Statische wie auch dynamische Eigenschaften der Softwarekomponenten werden im Abschnitt 5 - Design in geeigneten Diagrammen visualisiert und beschrieben. Kapitel 6 spricht die gesammelten Erfahrungen der vorherigen Entwicklungsphasen an. In 6.3 werden die Schwierigkeiten der Implementierungsphase beschrieben. Machbares, bzw. nicht Machbares, Wünschenswertes oder Vorhersagen sind in Kapitel 7 - Fazit und Ausblick.

2. Vergleichbarer Ansatz/Einordnung

2.1 Realisierungsmöglichkeiten im Hinblick auf geeignete Verzeichnisdienste

Für unser System ist außerdem relevant, die Ausleihstellen und daran geknüpfte zusätzliche Attribute ortsgebunden abzulegen. Mit der Möglichkeit Attribute ablegen zu können wird vor allem das Nachbilden eines sehr vereinfachten Verzeichnisdienstes für Webservices bezweckt. Ob diese Maßnahme ergriffen werden muss und ob diese realisierbar ist, wird in diesem Kapitel untersucht. Ein Verzeichnisdienst ist, vereinfacht gesagt, eine Art „Gelbe Seiten“, wo man nach bestimmten Diensten suchen kann. Hier sind es eben Webdienste, nach denen gegebenenfalls gesucht wird. In diesem Kapitel sollen die Eigenschaften einiger Verzeichnisdienste bzw. Systeme, die als solche dienen könnten, verglichen werden. Als Ergebnis dieses Vergleiches soll sich das für diese Diplomarbeit am besten geeignete System herauskristallisieren. Wie bereits in der Einleitung Kap.1 angedeutet, sollen Webservices registriert, gefunden und deren Schnittstellenbeschreibung gespeichert werden können. Da die einzelnen Dienste an bestimmte geographische Positionen gebunden werden sollen, um spezielle Funktionalität (s. Kap.4 – Analyse) anbieten zu können, muss der Verzeichnisdienst, geographische Daten verarbeiten können.

A. UDDI

UDDI steht für Universal Description, Discovery and Integration. Es ist ein Verzeichnis (Universal Business Registry) für verteilte webbasierte Informationen über Unternehmen oder Organisationen, die Webdienste anbieten wollen. Weiterhin ist es ein Framework für die Beschreibung, das Auffinden und das Publizieren von Webservices. Wegen letzterem erfüllt es die Anforderungen an einen Verzeichnisdienst für Webservices. Zu Beginn Sep.2000 wurde die Entwicklung von UDDI durch die führenden Softwarefirmen, wie Ariba, IBM und Microsoft überwacht und vorangetrieben. Diese waren für die Versionen 1.0 und 2.0 verantwortlich. Im Juli 2002 ist die dritte Version unter der Obhut von OASIS veröffentlicht worden. Letztendlich wurde die dritte Version am dritten Februar 2005 als Standard bestätigt.

Die OASIS behandelt die Standardisierung der Prozesse im Bereich von Webservices, also das Zusammenspiel verschiedener Webservices und die Integration in betriebswirtschaftliche Anwendungen. Dazu mehr im Kap. 2.2 über Webservices.

Arten von Informationsgewinnung in UDDI
Die UDDI Registry bietet drei Arten von Informationsgewinnung.
Die erste Art sind die white pages. Man findet einen Webservice indem man nach dem Unternehmen sucht, welches den jeweiligen Service anbietet. Die Kontaktinformationen sind z.B. Namen, Telefonnummern, Emailadressen oder Anschriften.
Die yellow pages ist die nächste Art der Informationsgewinnung. Hier werden die Geschäftseinheiten in verschiedene Klassen eingeteilt. Diese sind z.B. Branchen der Unternehmen, unterstützte Programmiersprachen, Laufzeitumgebungen oder Algorithmen für Webservices.
Die technische Seite von Webservices wird in den green pages dokumentiert. Dort erfährt man unter anderem, welches Kommunikationsprotokoll (wie z.B. SOAP) unterstützt wird und bekommt die Referenz auf die WSDL-Datei. Außerdem bekommt man hier Informationen über das Geschäftsmodell und die Geschäftsprozesse des jeweiligen Unternehmens.

UDDI Datenmodell
Das Datenmodell beschreibt die Zusammenhänge zwischen Organisationen, ihren Diensten und Schnittstellen. Die Abbildung 2.1.1 verdeutlicht dies.

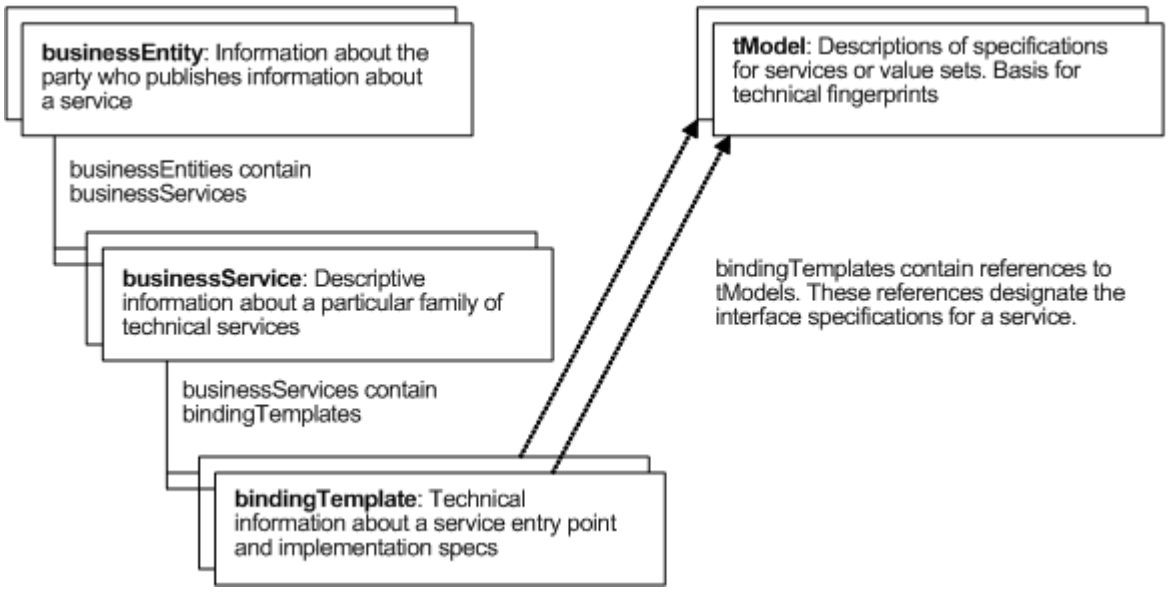

Abb. 2.1.1 Das Datenmodell von UDDI (wurde der UDDI-Spezifikation entnommen)

Business Entity enthält Informationen über den Anbieter, der Webservices anbietet.
Es enthält sowohl die Identifikations- als auch die Klassifikationsinformation. Obwohl eine Business Entity durch ihren businessKey innerhalb der Registry eindeutig identifizierbar ist, so gibt es noch weitere, branchenspezifische Identifikatoren wie z.B. eine Steuernummer. Diese sind im identifierBag gespeichert.
Business Service beschreibt eine Klasse verwandter Webservices eines Providers, einer Organisation bzw. eines Business Entity. Das kann z.B. ein Webservice sein, der in unterschiedlichen Versionen entwickelt wurde oder ein Service der in mehrere autonome Services unterteilt wurde. Business Service enthält mit categoryBag die Klassifikationsinformation.

Binding Template ist die Komponente, die einen einzigen Webservice repräsentiert. Das Template liefert die nötigen Informationen um einen Webservice anzusprechen und mit diesem interagieren zu können. Es enthält solche Informationen, wie die Adresse des Webservices bzw. dessen WSDL-Datei und die Referenzen auf tModels.
Die *tModels* (technical Model) befinden sich außerhalb der soeben beschriebenen Vererbungshierarchie, damit eine flexible technische Beschreibung eines Webservices möglich wird. Ein einzelnes tModel entspricht genau einer Ausprägung von Spezifikationen, transport- oder Kommunikationsprotokollen. Mit mehreren tModels, die an ein Binding Template gebunden sind, werden die technischen Eigenschaften eines Webservices bestimmt.
Mit Hilfe des *"Identifier Bag"* d.h. der Identifikationsinformation, des *„Category Bag"* d.h. der Klassifikationsinformation und des tModels können Dienste gefunden werden, die bestimmten Anforderungen genügen.

Bewertung des UDDI
Mit UDDI gibt es einen sehr ausgereiften Verzeichnisdienst, weshalb hier auch ein relativ komplexes Datenmodell zugrunde liegt. Diese Komplexität ist, im Bezug auf den Umgang mit UDDI, eher ein Hindernis und wird als ein Nachteil gewertet. Die hohe Komplexität verschafft UDDI jedoch auch Vorteile. Dadurch wird insbesondere die Flexibilität hinsichtlich der Unterstützung heterogener Systemlandschaften gefördert und eine hierarchische Suche ermöglicht. Der Verzeichnisdienst ist außerdem über das Internet allgemein verfügbar, was für eine Registry, auf dem globalen Markt der Webservices, von sehr hoher Bedeutung ist. UDDI ist nicht nur eine Registry, die Webservices verwaltet, sondern bietet auch dessen eigene Dienste als WSDL-Schnittstellen an. Dafür ist es nicht sehr weit verbreitet.
Die Anforderung an das System, den geographischen Bezug zu den Webservices bzw. zu den Businessentitäten, die diese verwalten herzustellen, wird nicht erfüllt.
Mit UDDI gibt es zumindest eine stabile standardisierte Grundlage eines Verzeichnisdienstes für Webservices. Aus diesem Grund wurde es hier aufgeführt und genauer erläutert, denn auf dieser Wissensgrundlage wird im Rahmen dieser Arbeit ein vereinfachter Verzeichnisdienst nachgebaut.

B. LBS

Die location-based services sind Dienste, die auf so genannten geographischen Informationssystemen (GIS) aufbauen, welche im nächsten Abschnitt vorgestellt werden. Die LBS werden für die Entwicklung von Notfallservices, Navigationssystemen, Tourenplanung für Touristen, "yellow maps" (eine Kombination von gelben Seiten und Landkarten), Routenverfolgung z.B. bei einem Autodiebstahl eingesetzt. Außerdem werden LBS für Umgebungssuchen und Routenberechnung eingesetzt. Für diese Diplomarbeit wird beabsichtigt die beiden letztgenannten Funktionalitäten für den Prototypen zu nutzen. In den Bereich der ortsgebundenen Dienste fällt das Lokalisieren eines Benutzers. Man nutzt dafür das GPS (global positioning system), welches für die Zwecke der Lokalisierung, die vorhandene Satelliteninfrastruktur nutzt. Die Benutzerposition auf der Erde wird mit Hilfe der GPS-Satelliten errechnet. Diese Eigenschaft wird in dieser Diplomarbeit nicht realisiert, doch könnte der Prototyp um diese zukünftig ergänzt werden.

Zur Gewährleistung der Interoperabilität zwischen LBS wurde das OGC gegründet, welches im Abschnitt für geographische Informationssysteme angesprochen wird.

Die LBS sind in verschiedene Kategorien eingeteilt. Diese sollen im Folgenden vorgestellt werden.

Personenbezogene (person oriented) LBS – Applikationen, die eine Lokalisierung einer Person nutzen.

Gerätebezogene (device oriented) LBS – Die Applikation kann, muss jedoch nicht auf eine Person fokussieren. Es kann ein Objekt oder eine Gruppe lokalisiert werden. Der bzw. das lokalisierte kontrolliert für gewöhnlich nicht den Service. Beispiel: Tracking eines gestohlenen Autos.

Pushservices- nach der Zustimmung des Empfängers (z.B. durch Installation eines entsprechenden Clientprogramms) können Nachrichten an diesen gesendet werden. Ein Beispiel dafür wäre eine Willkommensnachricht an einer Stadtgrenze. Ein Szenario mit höherer Priorität wäre eine Meldung über einen Terroranschlag in der näheren Umgebung.

Pullservices – Der Benutzer fordert eine Antwort an. Die Antwortinformation kann ortsgebunden sein. Beispiel dafür wäre eine Anfrage nach Cafes in naher Umgebung oder nach Bootsvermietungen bzw. deren Webservices, was bei dem zu entwickelnden System der Fall sein wird.

LBS –Einordnung der eigenen Anwendung
Die Anwendung dieses Diplomarbeitsprojektes wird in die Kategorie der Pullservices fallen, denn die Informationen werden durch den Nutzer des mobilen Gerätes beim ortsbasierten Webservice angefordert. Als Ausgangspunkt für die Suche in der Umgebung wird jedoch nicht die aktuelle Position des mobilen Gerätes verwendet. Stattdessen wird der Benutzer diesen Punkt selbst interaktiv bestimmen können. Da der Benutzer die Kontrolle über die Webserviceanforderungen behält, ist der zu entwickelnde Service personenorientiert. Außerdem wird es einerseits ein ortsbasierter Informationsservice (Kategorie: Information) sein, denn der Benutzer wird sich über die Boote, Dienstleistungen und die Umgebung informieren wollen. Andererseits, wegen der entgeltlichen Buchung von Booten, fällt der Service in den Bereich der ortsbasierten mobilen und kommerziellen Services (Kategorie: M-Commerce and Advertising).

Bewertung von LBS
Die LBS sind Dienste mit Ortsbezug. Nur wenige von denen bieten eine Kartendarstellung im Internet. Einer von denen, die es anbieten, ist der MapPoint-Webservice, der weiter unten genauer besprochen wird.

C. GIS

Im Folgenden werden wir uns den Geographischen Informationssystemen zuwenden, die daraufhin untersucht werden sollen, ob eine Realisierung eines ortsbezogenen Verzeichnisdienstes möglich ist.

Einführung in Geoinformationssysteme:
Die Verwendung von Satellitenaufnahmen aber auch der Luftbildfotografie (engl. aerial photography) machte es möglich häufige Veränderungen der Landschaften präzise und relativ schnell zu erfassen wie z.B. die Ausbreitung von Waldbränden, Überflutungen, Heuschreckenschwärme. Um die Bedeutung der erfassten Daten erkennen zu können mussten die Daten, nach entsprechenden Mustern, interpretiert und graphisch dargestellt werden. Dafür wurden spezielle "mapping tools", die s.g. geografischen Informationssysteme (GIS), entwickelt.

Die GIS sind Systeme, die Tools bereitstellen, welche zum Speichern, Wiedergewinnen, Transformieren und Darstellen von geographischen Daten dienen. Die Daten repräsentieren Naturereignisse der realen Welt. Repräsentiert wird die Position der Ereignisse im Bezug auf ein bekanntes Koordinatensystem, ihre positionsunabhängigen Attribute (wie z.B. Farbe, Auftreten von Krankheiten etc.), als auch die gegenseitigen Auswirkungen und Zusammenhänge der Ereignisse. Raumbezogene Daten sind in geographische und attributive Informationen untergliedert. Die geographischen Daten beschreiben z.B. die Lage, die Beziehungen zw. geometrischen Objekten wie z. B. Entfernungen, Flächengrößen usw. Die Merkmale oder Attribute beschreiben die Eigenschaften der Objekte, welche die geometrischen Daten repräsentieren. Ein Ankersymbol z.B., könnte einen Bootsverleih oder eine Anlegestelle der realen Welt entsprechen. Die zugehörigen Attribute wären dann z.B. der Name des Verleihs, die WSDL-URL der Dienstschnittstellenbeschreibung, oder auch Attribute, welche die Dienstleistung beschreiben, wie z.B. die Art von Angeboten.

Aufgaben geographischer Informationssysteme
Die Aufgaben der Software eines GIS werden in vier Komponenten eingeteilt. Die Aufgabenbereiche der GIS-Komponenten Eingabe, Verwaltung, Analyse und Präsentation werden im Folgenden kurz erläutert.

Erfassung raumbezogener Daten
Hier wird eine Datenbasis eines GIS geschaffen. Vor der Erfassung von Daten findet die Datenmodellierung statt. Vektordaten werden durch Geländeaufnahmen, Luftbildmessung oder Digitalisierung erfasst. Sachdaten können durch alphanumerische Eingabe erfasst werden, aus Papierakten, aus bereits existierenden Datenbanken oder Tabellenkalkulationen übernommen werden. Rasterdaten können vom Satelliten aufgenommen oder z.B. gescannt werden. Transformationen in ein einheitliches Bezugssystem, Datenkonversionen zur Integration von Rasterdaten in Vektorsysteme und umgekehrt sowie die Objekterzeugung sind für die Integration der Daten in das Datenmodell notwendig.

Verwaltung raumbezogener Daten
Die vorher erfassten Daten werden in einer Datenbank gehalten. Außer der Datenhaltung gehört zur Verwaltung der Daten, sowohl das Manipulieren der Daten über Grundfunktionen wie Verschieben, Drehen, Löschen, Aufteilen und Verschmelzen, sowie die Umklassifizierung und der Datenaustausch mit anderen Systemen. Dabei können die dynamischen Vorgänge, wie Drehen, Verschieben eher bei Trackingsystemen (s.LBS) angetroffen werden. Auf der Datenverwaltungsseite wird über geeignete Datenformate entschieden. Die Eignungskriterien sind das Datenvolumen und die Zugriffsgeschwindigkeit. Für die Daten müssen durch Datenmodellierung geeignete Strukturen festgelegt werden. Dabei wird ein **geometrisches** (bezogen auf die Position eines Objektes), ein topologisches (innerer Aufbau eines Objektes beschrieben durch Knoten und Kanten) und ein **thematisches** (auf die Semantik des Objektes bezogenes) Modell erstellt. Beim Aufbau des Verleihsystems wird sowohl das geometrische als auch das thematische Modell direkt vom Entwickler gestaltet bzw. beeinflusst oder sogar vom Nutzer des Systems mitgestaltet. Beispielsweise wird ein Bootsverleihsymbol durch die Angabe einer Adresse auf der Karte platziert(geometrisches Modell) und die Beschreibung der Dienstleistungsart des Verleihs, anhand von Attributen, vom Verleiher selbst abgelegt. Auf das topologische Modell hat der Entwickler keinen direkten Einfluss. Nach der Datenmodellierung muss noch ein Datenbankmodell erstellt und ein entsprechendes DBMS gewählt werden.

Analyse raumbezogener Daten
Bei der Analyse werden aus vorhandenen Daten neue Informationen gewonnen. Diese dienen als Entscheidungsgrundlagen. Dafür werden Operationen angewandt die geometrische, logische oder relationale Verknüpfungen der Daten durchführen lassen. Auch statistische Verfahren können genutzt werden. Zusätzlich zu einer Datenbank wird eine so genannte Methodenbank geführt, die alle relevanten Algorithmen verwaltet.

Präsentation raumbezogener Daten
Eine geeignete Visualisierung der Informationen ist für die Bennutzerakzeptanz eines Systems von großer Bedeutung. Eine grafische Darstellung gegenüber einer Alphanumerischen verbessert oft das Verständnis eines bestimmten Sachverhalts. Bei der Ausgabekomponente geht es deshalb unter anderem um die geeigneten Präsentations- und Visualisierungstechniken. Bei heutigen GIS wird unter anderem auch das Internet als Präsentationsmedium genutzt (s. WebGIS weiter Unten bei der Systemkategorisierung). Als Präsentationsmedium des zu entwickelnden Systems, wird ein mobiles Gerät dienen.

Einordnung der GIS-Aufgaben in das eigene Systemmodell
Für das Bootsverleisystem wird nach einem GIS gesucht, das einen Bestand an geographischen Daten bereits vorweisen kann. Dieser Bestand, muss durch den Benutzer erweitert werden können. Zumindest eine Erweiterung um neue POIs, welche die Bootsverleihe repräsentieren, muss realisierbar sein. Beim Registrieren eines Webservices wird ein Bootsprovider den Webservice geocodiert hinterlegen, dabei sollte das GIS die geocodierung übernehmen. Zusätzlich werden einige, den Dienst beschreibende, Sachdaten in Form von Attributen hinterlegt. Die Analyse wird das Anbieten von Diensten wie die Routenberechnung oder Umgebungssuche ermöglichen. Das Anzeigen der gewonnenen Informationen wird in den Aufgabenbereich der Präsentationskomponente fallen, welche eine geeignete Kartenausgabe für das mobile Gerät vorbereitet.

Ein GIS sollte gewisse grundsätzliche Anforderungen erfüllen, die im Folgenden aufgelistet werden.

GIS – Ausprägungen:
Auf Grund der Vielfalt der Anwendungen sind im Laufe der Zeit in den einzelnen Fachdisziplinen eigene Geo-Iformationssysteme (s. Abb.)entstanden.

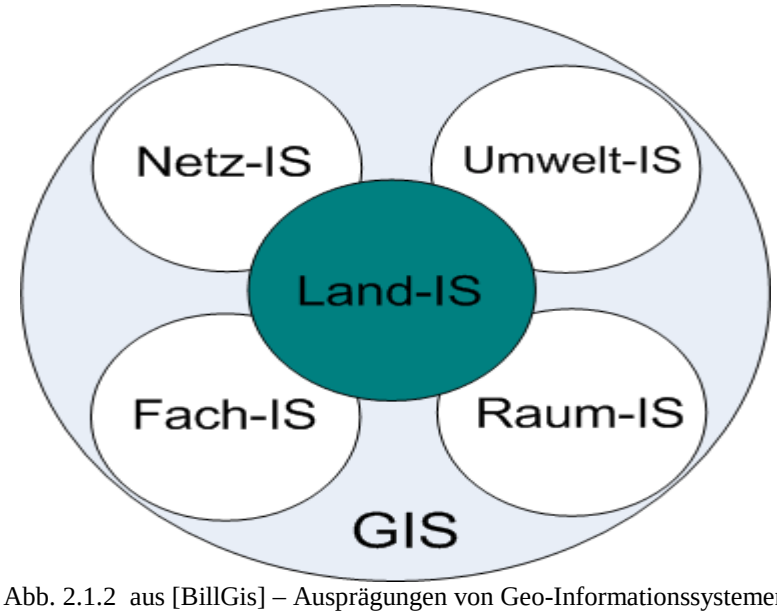

Abb. 2.1.2 aus [BillGis] – Ausprägungen von Geo-Informationssystemen

- So übernehmen **Landinformationssysteme(LIS)** z.B. die exakte geometrische Erfassung des Grund und Bodens einer Region, sowie hiermit verknüpfte Sachdaten. Geländeoberflächen, Luft- und Satellitenbilder können topografisch beschrieben und der Raumbezug durch Koordinaten hergestellt werden. Die Aufgabengebiete der LIS sind in erster Linie das Vermessungswesen und die Liegenschaftskataster (Liegenschaftskarte und Liegenschaftsbuch). Diese Systeme dienen als Basis für alle weiteren, die nachfolgend aufgeführt werden.
- Für Geographen, Raumplaner und Statistiker haben sich **Rauminformationssysteme (RIS)** als Instrumente zur Entscheidungsfindung und als Hilfsmittel für Planung und Entwicklung herausgebildet. Zu den Aufgaben solcher Systeme gehört unter anderem die Erfassung, Verwaltung, Analyse und Präsentation der Bevölkerungs-, wirtschafts- und Siedlungsentwicklung. Die Grundlage bildet der einheitliche Raumbezug, der durch die Nutzung von LIS gegeben ist. Zusätzlich werden sekundäre Metriken (flächenbezogene Angaben - s. Glossar) benutzt und überwiegend thematische Daten(Sachdaten) behandelt.
- **Umweltinformationssysteme(UIS)** sind erweiterte GIS die, sich mit dem Zustand der Umwelt befassen. Die Daten eines solchen UIS beschreiben den Zustand der Umwelt im Hinblick auf Belastungen und Gefährdungen. Diese Systeme dienen in erster Linie als Grundlage zum Schaffen von Maßnahmen für den Umweltschutz. Zu den Anwendungen von UIS gehört die Erfassung von Luft-, Boden- und Wasserqualität, Radioaktivität, Biotopkartierungen als auch die Erhaltung der Artenvielfalt, Feststellung von Pflanzenschäden und Gesundheitsrisiken. Die UIS können, auf Grund der besonderen Bedeutung des zeitlichen Ablaufs, als dynamische Systeme bezeichnet werden, wobei die anderen Systemausprägungen eher statische Daten führen.
- **Netzinformationssysteme(NIS)** gehören zu den meist genutzten GIS. Sie dokumentieren und bearbeiten die Betriebsmitteldaten d.h. z.B. Kundendaten, Leitungen oder Anlagen zur Ver- und Entsorgung.
- **Fachinformationssysteme(FIS) –** Jedes der bisher vorgestellten GIS-Ausprägungen reiht sich auch bei den Fachinformationssystem ein, doch es gibt noch weitere Sonderformen von GIS, die durch die bereits vorgestellten Ausprägungen gar nicht oder nur teilweise abgedeckt werden. Zu dieser Kategorie zählt die Telekommunikation und auch der Verkehrs- und Logistiksektor. Derartige Systeme müssen Aufgaben wie Navigation oder Standortbestimmung lösen.

Einordnung des Bootsverleihs bei den Ausprägungen von GIS
Das GIS, welches für die Entwicklung des ortsbasierten Bootsverleihsystems verwendet werden soll, wird sich bei den fachspezifischen Fachinformationssystemen einreihen und wie alle anderen GIS-Ausprägungen, die Landinformationssysteme als ein Basissystem nutzen.

GIS - Kategorisierung:
Die hier vorgenommene Kategorisierung richtet sich nach dem Buch [BillGIS] eines Hard- und Softwareentwicklungsingenieur im GIS-Bereich. Er kategorisiert die Systeme wie folgt.

Universelle Geo-Informationssysteme bzw. High end GIS. Systeme wie Arc/Info, Sicad/open, Smallworld GIS oder Microstation GIS Environment werden hier als Beispiele angefügt. Diese Systeme bieten das volle Spektrum an Funktionalität bezüglich der Erfassung, Verwaltung, Analyse und Präsentation ortsbezogener Daten. Sie sind allerdings relativ teuer (der Preis liegt oft über 50.000 EUR). Außerdem benötigen sie einen hohen Einarbeitungsaufwand. Große Datenmengen können bereitgestellt und im Multiuserbetrieb

zugegriffen werden. Eingesetzt werden sie z.B. von Ver- und Entsorgungsunternehmen, von Vermessungsfirmen und in der Kommunalverwaltung. Mit ArcInfo z.B. ist ein GIS der Firma Environmental Systems Research Institute (ESRI) gemeint. Eine interessante Beispielanwendung, welche auf ArcInfo aufsetzt ist der Gesundheitsatlas der Niederlande[Zorg]. Der Atlas wurde in [WebMap] genauer beschrieben. Beim SmallWorld GIS liegt der Schwerpunkt auf Attributierung. Zu Beginn wurde das GIS in Smalltalk entwickelt.

Low-cost GIS oder PC-GIS bieten mittlere Datenmengen an, auf die meistens im Singleusermode zugegriffen werden kann. Sie decken meistens nur einen konkreten Anwendungsbereich ab. Dafür wird fast das komplette Funktionalitätenspektrum angeboten und sie sind günstiger als die universellen GIS. Der Preis liegt meistens unter 10.000 EUR. Beispielprodukte sind unter anderem PC ARC/INFO, Atlas*GIS und MapInfo.

Desktop-GIS, wie z.B. ArcView von ESRI, Sicad Spatial Desktop oder GeoMedia, sind Systeme, die mit Präsentationsfunktionalitäten meistens gut ausgestattet sind. Im Bezug auf Datenerfassung, Verwaltung und Analyse sind die Systeme oft eingeschränkt. Sie sind nicht multiuserfähig und verfügen kaum über AnwendungsschalenDie Kosten für ein derartiges GIS belaufen sich auf nicht mehr als 5.000EUR.

Internet-GIS bzw. WebGIS bilden eine Klasse von GIS, deren Stärken, ähnlich wie bei den Desktop-GIS in der Präsentation liegen. Was die Bezeichnung bereits vermuten lässt, sind diese Systeme über das Internet erreichbar und erlauben damit, dass sich über den Sachbearbeiter hinaus durchaus auch interessierte Gelegenheitsnutzer angesprochen fühlen können. Diese Systeme bewegen sich in der niedrigsten Preisklasse von GIS. Sie sind für ein paar 100 bis ein paar 1000 EUR zu erwerben und werden repräsentiert von Produkten wie z.B. ArcView Internet Map Server, AutoDesk Map Guide und GeoMedia Webmap.

Auskunftsysteme präsentieren raumbezogene Daten. Sachbearbeiter in Kommunalverwaltungen nutzen diese für diverse Auskünfte wie z.B. Lageplanauskünfte. Die Systeme sind meistens relativ leicht bedienbar und liegen in der gleichen Preisklasse wie die eben besprochenen Internet-GIS. Die typischen Repräsentanten sind z.B. DAVID-Auskunft, AKSYS und visor.

Spezialisierte GIS stellen, an spezielle Anwendungen angepasste Funktionspakete bereit oder überlassen dem Benutzer die Zusammenstellung. Kanalinformationssysteme wie z.B. SICAD-KANDIS und InKa gehören zu dieser Klasse. Aus dem Bereich der Straßendaten mit Routensuche und Touristeninformation ist z.B. das CardyTSP bekannt. Die Preise solcher Systeme variieren stark und bewegen sich innerhalb aller Preisklassen von geografischen Informationssystemen. Die niedrigste Preisklasse bilden auch hier die Massenmarktpreise.

Die GIS- bzw. Geodaten-Server oder Mapserver enthalten Funktionalitäten zur Verwaltung und den Zugriff auf große Datenmengen. Auch eingeschränkte GIS-Funktionen können angeboten werden. Sie treten auch als Kopplung zw. Webserver und Geodatenbank auf. Beispielsysteme sind, ArcView Internet Map Server, GeoMediaWeb Server, Spatial Database Engine und einige mehr. Die Kosten liegen bei mehreren Tausend Euro.

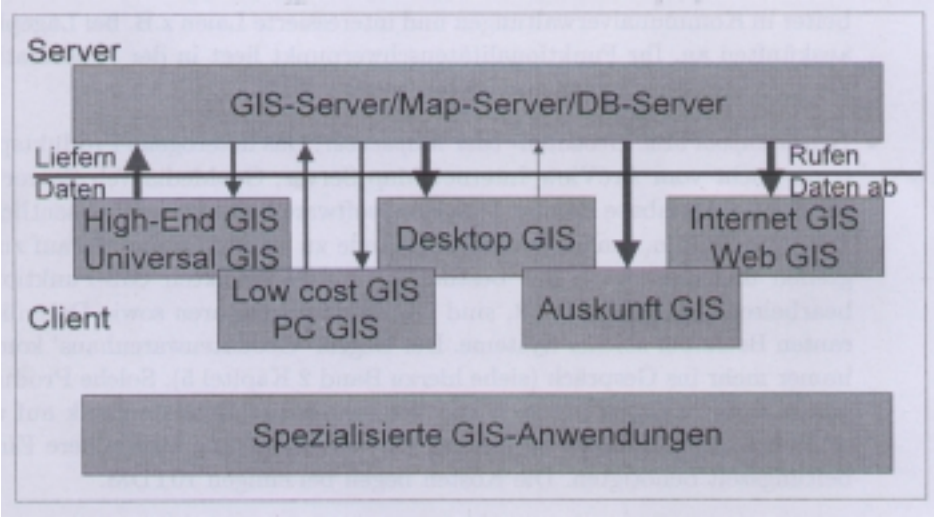

Abb. 2.1.3 aus [BillGis] – Systemkategorien im Zusammenspiel

Bewertung geographischer Informationssysteme

Die zuerst vorgestellte Kategorie von GIS (High-End GIS s.Abb.) ist wohl die mächtigste im Bezug auf deren Umfang an angebotenen Funktionalitäten. Viele GIS-spezifische Anwendungsbereiche wurden berücksichtigt und in so genannten Anwendungsschalen angeboten. Diese Systeme bewegen sich dafür in der höchsten Preisklasse. Bei den GIS der anderen Kategorien nimmt man Einschränkungen der Funktionalität und beim Datenvolumen in Kauf um die Systeme günstiger anbieten zu können. Dadurch sind diese jedoch auch besser zu warten und weniger schulungsintensiv. Diese Entwicklung ist wohl darauf zurückzuführen, dass sich die potenzielle Klientel der GIS verändert bzw. erweitert. Wo in den Anfängen von GIS lediglich hoch qualifizierte Sachbearbeiter aus dem Vermessungswesen, aus der Umwelt-, Planungs- und Kommunalverwaltungen oder Ingenieurbüros mit GIS gearbeitet haben, so sind heute auch Gelegenheitsnutzer an speziellen GIS-Diensten, wie z.B. der beabsichtigte ortsbezogene Verzeichnisdienst. Navigationssysteme, ortsabhängige Dienste (s. nächster Abschnitt „LBS“) wie Umgebungssuche oder Routenberechnung liegen heute im Trend.

Heutige GIS haben meist einen modularen Aufbau und graphische Benutzeroberflächen, die deren Handhabung vereinfachen. Dennoch werden zur Anwendung und Bedienung intensiv geschulte Spezialisten benötigt und die Anschaffungskosten sind relativ hoch. Außerdem wird für den Aufbau des Bootsverleihs ein GIS mit einer Webservice-Schnittstelle benötigt, die über das Internet verfügbar ist. Dies gibt es bei den konventionellen GIS bisher nicht. Die Datenmodelle konventioneller GIS-Systeme sind nicht darauf ausgerichtet, benutzerspezifische Datenbanken zu verwalten. Deshalb wäre das Ablegen der Bootsproviderdienste nicht möglich bzw. sehr schwierig. Die nötigen ortsbezogenen Funktionalitäten, die für den ortsbasierten Verzeichnisdienst benötig werden, sind insbesondere bei den Web-GIS vorhanden, doch gibt es auch hier keine Webserviceschnittstellen und das Ablegen benutzerdefinierter Daten ist nicht möglich.

Das OGC:

Das Internet ist ein wunderbares Medium, um Dienste verschiedener Art einer fast unerschöpflichen Menge an Nutzern anbieten zu können. Das **Open Geospatial Consortium** (OGC) beschäftigt sich mit dem Entwurf von Standards wie dem „OpenGIS Location Services“ (OpenLS), die eine bessere Nutzung von GIS über das Internet ermöglichen sollen. Das Ziel ist das Erreichen möglichst hoher Interoperabilität von Geo-Informationssystemen.

Durch solche Standardisierungen gibt es fest definierte Schnittstellen, die eine reibungslose Zusammenarbeit der verschiedenen Netzdienste ermöglichen. Monolithische und proprietäre

Systeme sollen durch Komponenten mit Standardisierten Schnittstellen ersetzt werden. An der Organisation beteiligen sich verschiedene IT-Firmen, GIS-Hersteller, Datenbankhersteller, Datenprovider, Dienstleister und Universitäten der Welt. Die Mitglieder verpflichten sich die Spezifizierten Schnittstellen in ihren eigenen Produkten einzubauen, so dass im Laufe der Zeit immer mehr OGC konforme GIS entstehen und eine Interoperabilität möglich wird.

D. MapPoint

Wie bereits festgestellt wurde, brauchen wir, um die Anforderungen bezüglich der Behandlung von ortsabhängigen Daten erfüllen zu können, so etwas wie ein geographisches Informationssystem (GIS s. Kap. 2.1.B) , welches über eine Webserviceschnittstelle zugänglich ist. Damit streben wir an, so etwas wie die im Abschnitt B. beschriebenen Pullservices, den Nutzern unseres Systems anbieten zu können. Geographische Systeme sind jedoch meist relativ teuer und wegen ihrer Komplexität schwer zu erlernen. Es gibt bisher auch nur wenige LBS, die eine Kartendarstellung über das Internet anbieten. Deswegen wird in diesem Kapitel, eine, weniger komplexe und doch sehr ausgereifte Version eines GIS vorgestellt. Es ist der MapPoint-Server von Microsoft.

Microsofts Mappoint Server

Der Server bietet eine Webserviceschnittstelle, die den Entwicklern die Integration von Standort-basierten Dienstleistungen ermöglicht. Es lassen sich damit beispielsweise Karten-, Fahrtrichtungs- oder Umgebungssuchen in die entwickelten Anwendungen oder Geschäftsprozesse integrieren. Der MapPoint Server wird bei Microsoft gehostet und enthält deshalb bereits einige Datenbestände. Trotzdem ist es natürlich auch möglich eigene Datenbestände hoch zu laden und an eine geographische Position zu binden. Dank dieser Fähigkeiten des MapPoint-Servers ist wohl die wichtigste Voraussetzung bezüglich eines „geographischen Verzeichnisdienstes“ erfüllt. Dadurch können nämlich eigene Webservices ortsgebunden abgelegt werden. Diese werden jedoch nicht vom Systembetreiber publiziert, sondern vom jeweiligen Bootsprovider. Dafür müsste jeder Provider einen eigenen Account erhalten. Es könnte ausreichen, die Provider bei unserem System registrieren zu lassen und die Nutzung des MapPoint-Webservices über den allgemeinen Account des Systems gewähren. Man kann aber auch die Credentials des MapPoint-Servers nutzen und dadurch einem Bootsprovider einen Login direkt beim MapPoint-Server ermöglichen.
Die Datenbestände (s. Literaturliste „MapPointWebserviceDataSources“) des Servers umfassen Daten, die über spezielle Funktionen genutzt werden können. Damit können Landkarten gezeichnet, Orte und Adressen gefunden und Routen berechnet werden. Dem MapPoint Webservice liegt eine Datenbank zu Grunde die mehr als 15mio. POIs (points of interest) enthält und sie wird ständig durch neue erweitert.

Benutzerdefinierte Orte

Wie bereits erwähnt wurde, ist ein relativ einfaches Geokodieren und Sichern kundenspezifischer Datenbanken möglich. Falls keine Breiten- und Längen-Angaben übergeben wurden, was wahrscheinlich meistens der Fall sein wird, wird automatisch geokodiert. Bei diesem Vorgang, werden den in der benutzerspezifischen Datenbank abgelegten Entities, auch POIs genannt, aus der angegebenen Adressinformation, die entsprechenden Weltkoordinaten zugeordnet. Die so gespeicherten Daten können anschließend von MapPoint-Webservice-Applikationen verwendet werden. Das Hochladen eigener Datenbestände geschieht über einen anderen Dienst des MapPoint-Servers, den Customer Data Service. Eigene POIs, mit ihren Ortsdaten und Symbolen werden in eigener DB auf dem MS-Server gehalten. Zu den POIs können 300 Attribute spezifiziert werden.

Programmierschnittstellen, wie die Microsoft-Webservice-APIs fürs Suchen und Anzeigen, sind vorhanden.
Im Folgenden werden die wichtigsten Eigenschaften/Komponenten des Mappoint Webservice vorgestellt. Insbesondere wird hier auf verschiedene Arten der Suche nach bestimmten Daten des geografischen Systems eingegangen.

Suche nach bestimmten Attributen
Unternehmen können um die 300 Attribute, solche wie die Dienstart oder Öffnungszeiten, spezifizieren und mit abspeichern. Im Fall der Bootsverleihe wird sich die Attributsuche auf die einzelnen Bootsverleihe beziehen. Damit ist der Bootsverleiher in der Lage die Art seiner Dienste bereits bei der Suche berücksichtigen zu lassen. Dem Benutzer des Verzeichnisdienstes wird diese Art der Suche erlauben nach Bootsverleihen zu suchen, die z.B. früh morgens aufmachen oder die eine bestimmte Art von Booten vermieten. Es können also unbrauchbare Dienste, bereits bei der Suche aussortiert werden und nicht erst nach dem Zugriff auf den jeweiligen Webservice. Um die wichtigste Fähigkeit eines Webservice-Verzeichnisdienstes wie z.B. UDDI nachbilden zu können, wird es außerdem wichtig sein eine URL als Attribut ablegen zu können, denn wir müssen die Webservices bzw. die entsprechenden WSDL-Beschreibungen[2] der Bootsverleihstellen referenzieren können. Genau das ermöglicht MapPoint durch die Attributspezifikation zu den POIs. Eine andere Möglichkeit wäre es, eine WSDL-Datei als Attribut abzulegen.

Suchen nach Adressen
Suchen nach geokodierten Adressen in Nord Amerika, Süd Amerika und Europa. Man sucht also nach einer bestimmten Stelle auf der Erdoberfläche (obwohl außerdem auch Monddaten vorliegen). In den Karten von Europa sind noch nicht alle Länder komplett enthalten. Die Karten von Deutschland sind jedoch relativ vollständig. Straßennamen lassen sich mit dem Namen der Stadt, dem Namen des Landes, Kreuzungen und der Postleitzahl kombinieren, um ein bestimmtes Ergebnis zu erhalten. Im Bezug auf den zu simulierenden „geographischen Verzeichnisdienst“, könnte ein Benutzer nach einer Adresse suchen und anschließend eine Umgebungssuche starten, um in der Umgebung der Adresse alle dort befindlichen Webservices, bzw. Webservices die, die Bootsverleihstellen repräsentieren, aufzulisten. Die Suche nach einem bestimmten Bootsvermieter könnte damit ebenfalls realisiert werden. Dafür müsste jedoch die Adresse dem Benutzer bekannt sein und dies wird eher selten der Fall sein.

Suchen nicht adressierbarer Stellen
MapPoint bietet noch eine semantische Suche an. Es ist die Suche nach geographischen Entitäten, solchen wie Städte, Postleitzahlen, Länder, Landkreise, Flüsse, Seen, Flughäfen, Landmarken, und anderen Strukturen die nicht durch Straßenangeben zu finden sind. Mit dieser Funktionalität bekommen wir ein Werkzeug in die Hand, das uns eine Regionalsuche ermöglicht. Wir können somit nach Bootsverleih-Webservices einer bestimmten Region, wie z.B. Hamburg oder deren Stadtteilen, suchen. Diese Art der Suche scheint die am meisten geeignete für Verleihservices zu sein. Vor allem, weil nicht davon auszugehen ist, dass die genaue Adresse dem Benutzer bekannt sein wird.

Finden von Plätzen in naher Umgebung
Als Ergebnis einer derartigen Suchaktion bekommt man eine Liste mit POIs (points of interest) aus der Nachbarschaft einer ausgewählten Stelle. Man kann damit beispielsweise eine Nachbarschaftssuche entlang einer festgelegten Route starten. Hotels oder Restaurants können ihren Gästen z.B. die Positionen der nahe liegenden Geldautomaten anbieten. Die

[2] WSDL – WebService Description Language

Kunden eines Autohändlers könnten die Lagen aller, in der Umgebung befindlicher, Tankstellen benötigen. Wir werden diese Funktionalität jedoch dafür nutzen, um eine Suche nach Bootsverleihen in der Umgebung einer bestimmten Adresse, dem Nutzer unseres Systems anzubieten.

Reverse Geocoding
Mit Reverse Geocoding bietet MapPoint eine weitere Art der Suche. Es handelt sich dabei um den Zugriff auf eine Liste geographischer Entitäten, wie Landkreise(counties) oder designierte Verkaufsregionen, die durch Angabe geeigneter Koordinaten assoziiert werden. Da wir nach ganz bestimmten POIs suchen wollen, wird diese art der Suche bei der Entwicklung des Prototypen nicht berücksichtigt.

Routing
MapPoints Routing ist wiederum eine Funktionalität, die sehr gut zu unserem System passt.
Firmen können ihren Kunden eine Wegbeschreibung liefern, denn MapPoint unterstützt eine optimierte Fahrtbeschreibung mit Instruktionen Schritt für Schritt. Sowohl eine grafische als auch eine schriftliche Beschreibung steht dem Benutzer bzw. dem Programmierer zur Verfügung.

2.2 Webservices – Vorteile im Vergleich zu anderen Middlewaresystemen

Hier werden Webservices grob und allgemein vorgestellt, d.h. der Leser erfährt wie Webservices definiert werden, wie die WS-Architektur aufgebaut ist und wie der Einsatz von Webservices motiviert werden kann. Mögliche Szenarien zum Aufbau einer WS-Umgebung vor der eigentlichen Interaktion sollen hier auch nicht fehlen. Anschließend werden die Vorteile von Webservices gegenüber der bisher bekannten Middlewaresysteme gezeigt. Die wohl wichtigste Funktion einer Middleware wird von Webservices erfüllt, nämlich die Abstraktion von der Netzwerkprogrammierung. Vor zwei oder drei Jahren wäre ein mobiles System auf der Basis von Webservices undenkbar. Zumal Webservices eine ganz junge Technologie ist, aber auch die Fähigkeiten mobiler Geräte waren nicht die gleichen wie wir sie heute haben. Die damals verwendete Technologie war das WAP (Wireless Application Protocol). Es ist ein speziell für Handy-Displays ausgelegter Übertragungs- und Darstellungsstandard. Es baut auf den bewährten Internetstandards, wie IP, URLs und XML, auf. Für WAP-Clients gibt es eine eigene XML basierte Markupsprache, die Wireless Markup Language (WML). Sie ermöglicht das darstellen von Inhalten auf den Displays mobiler Geräte. WAP wird Heute immer noch eingesetzt, jedoch verliert es, mit dem immer stärkeren Aufkommen von Smartphones und einwahlfähigen PDAs mit relativ großen Displays, an Attraktivität. In dieser Arbeit werden mobile Geräte besonders fokussiert, deshalb werden auch im Bereich der Webservices Systeme angeschaut, die zumindest die Entwicklung eines mobilen Webserviceclients ermöglichen. Diese Systeme werden jedoch erst im Kapitel über mobile Geräte (s. Kap. 3) betrachtet.

A. Webservices – Einführung

Webservices – Definition
Ein Web Service ist eine, durch eine URI identifizierbare, Applikation deren Schnittstellen und Bindungen an Protokolle mittels XML-Artefakten beschrieben werden können. Ein Web Service kommuniziert direkt mit anderen Softwareagenten über XML basierte Nachrichten, die durch Internet basierte Protokolle ausgetauscht werden.

Leistungen von Webservices
Ein grober Überblick über die Leistungen von Webservices soll an dieser Stelle gegeben werden. Auf die einzelnen Komponenten wird etwas weiter unten, bei der Webservice-Architekturbeschreibung, Bezug genommen.
Durch die Verwendung von SOAP wird von der Kommunikationsschicht abstrahiert. Das SOAP Messaging Interaktionsmodell sorgt für den Nachrichtenaustausch zwischen zwei Endpoints. Dabei werden die Daten mithilfe von XML-Parsing aus den SOAP-Nachrichten extrahiert. Das Umwandeln der Daten in ein sprachunabhängiges Format, nennt man Serialisieren und den umgekehrten Prozess deserialisieren. Die Nutzung von RPC (De)Serialisierung, SOAP, WSDL und XML ermöglicht somit die Unabhängigkeit von programmiersprachlichen Typen.In der Regel wird das HTTP als Transportprotokoll genutzt. Falls jedoch eine asynchrone Kommunikation benötigt wird, kann JMS oder SMTP verwendet werden. Eine Entkopplung von Service und Ort wird durch WSDL und UDDI gewährleistet. D.h. Ein bestimmter Webservice wird nur bei Bedarf angesprochen. Dieser kann, muss aber nicht im System fest integriert sein. Er kann von einem beliebigen Anbieter stammen und sich an einem beliebigen Ort im Netzwerk befinden. Die Benutzung des weltweiten Internet macht die Dimension nahezu unendlich. Die WSDL spielt im Bereich der Webservices eine tragende Rolle. Sie ist eine Beschreibungssprache, die für eine Entkopplung von Programmiersprachen und Plattformen der beteiligten Geräte, sorgt. Ein WSDL-Dokument beschreibt, mithilfe der WSDL, den gesamten Webservice.

Die ***Architektur von Webservices*** ist eine Schichtenarchitektur und wurde von W3C[3] bzw. von der Webservices Architecture Working Group des W3C entwickelt.
Die Abbildung zeigt die, von der Gruppe spezifizierte Schichtenarchitektur.

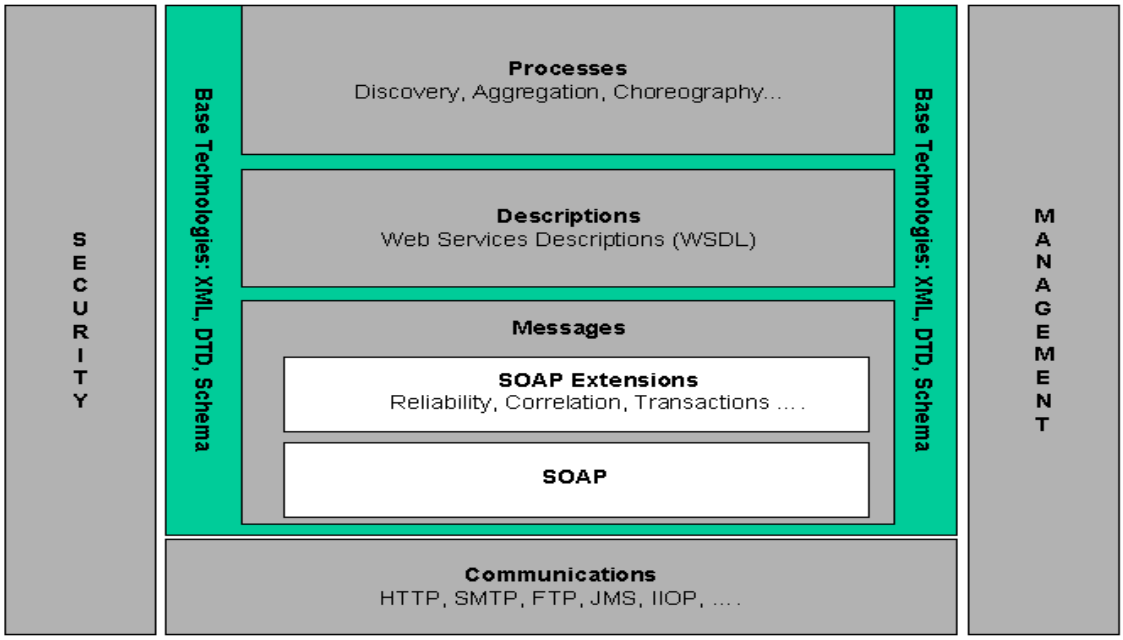

Abb. 2.2.1 Die Webservices-Schichtenarchitektur von W3C

Die Wahl von **XML** als die Sprache für Webservices, als auch die Verwendung der heute gängigen Transportprotokolle des Internet als Transportprotokolle für Webservices waren kluge Entscheidungen des W3C. Diese Entscheidungen haben sehr dazu beigetragen, dass sich Webservices am Markt durchsetzen können. XML ist die zentrale Beschreibungssprache des Internet und die Sprache zur Beschreibung von Webservices.

[3] World Wide Web Consortium

Die Nachrichten, die mithilfe der Internettransportprotokolle transportiert werden, werden in ihrer Struktur über **SOAP** (Simple Object Access Protocol) beschrieben. Diese Beschreibung ist unabhängig von der Anwendung. D.h. sie ist unabhängig von der Programmiersprache der Anwendung. Die Basis des SOAP ist einfach gehalten und umfasst nicht Gebiete wie Sicherheit, Verlässlichkeit d.h. die Dienstgüte, Transaktionen oder das Routen von Nachrichten. Doch gibt es die Möglichkeit, mit so genannten **SOAP-Extensions** in den Verarbeitungsprozess von SOAP-Anfragen/-Antworten einzugreifen und eigene Erweiterungen zu hinterlegen. Wie aus der obigen Abbildung ersichtlich, kann SOAP an alle gängigen Transportprotokolle gebunden werden. Meistens wird jedoch das **HTTP** (Hyper Text Transport Protokoll) verwendet. Wahrscheinlich wegen des Request-Response-Verhaltens, das bei dem RPC-Encoding ebenso statt findet. RPC-Encoding ist eine bestimmte Art SOAP-Nachrichten zu kodieren. RPC steht für Remote Procedure Call und wird weiter unten detaillierter behandelt. Um weitere Informationen zu den eben genannten Begrifflichkeiten zu erhalten verweise ich auf weiterführende Literatur.
Die nächst höhere Schicht über der SOAP-Schicht ist die Description(Beschreibungs)-Schicht. Es handelt sich hier um eine formale Beschreibung eines Webservices. Alles was sich formal beschreiben lässt, ist automatisierbar. So kann die Software des Dienstnutzers als auch die des Diensterbringers automatisch erstellt werden, sobald alle nötigen Schnittstellen und Ein/Ausgabe Parameter definiert wurden. Außerdem wird man in einer **WSDL**-Datei eine formale Beschreibung der Nachrichten des Dienstes, der Bindungen an Transportprotokolle finden und man findet Adressen hinter denen sich die tatsächlichen Webservices verbergen. Die oberste Schicht der Webservicearchitektur beschäftigt sich mit Prozessen. Damit sind **Prozesse** gemeint, wie z.B. die Entdeckung von Webservices (durch Dienste wie: **UDDI** oder Web Service **Inspection**). Die Betrachtung von Sicherheitsmechanismen bei Webservices unterscheidet sich etwas von der Betrachtung anderer Internettechnologien. Viren können, als Webservice beschrieben, trotz Firewalls auf den Rechner gelangen. Diese Betrachtung wird, wegen ihrer Relevanz, bereits in der Architektur schichtenübergreifend berücksichtigt. Ein entsprechendes Sicherheitsmodell wird durch die WS-Security Spezifikation beschrieben. Außerdem gibt es noch weitere sechs Spezifikationen.

Eines der Vorzüge dieser Architektur von Webservices ist die saubere Integration in die Webarchitektur. WS sind nämlich als Internetressourcen adressierbar, sie können unter anderem mit Proxies zusammenarbeiten. Die Architektur ermöglicht Plattformunabhängigkeit, da sie keine bestimmte Art der Interaktion zwischen Komponenten vorschreibt. Sie umfaßt Aspekte der Sicherheit und Verlässlichkeit. Außerdem setzt sie XML für alle beteiligten Technologien als Beschreibungssprache voraus. Sie ist relativ einfach und modular aufgebaut. Außerdem wird eine Vielzahl an Kommunikationsprotokollen unterstützt, was die Erweiterbarkeit erhöht.

Das Rollenmodell von Webservices sieht drei Rollen vor. Es gibt den Diensterbringer, den Dienstnutzer und den Vermittler. Der Diensterbringer wird seinen Service wahrscheinlich beim Verzeichnisdienst anmelden bzw. publizieren damit der Service überhaupt auffindbar wird und der Dienstnutzer sich die gewünschte Servicebeschreibung vom Verzeichnisdienst besorgen kann um seine Clientanwendung entsprechend der WSDL aufbauen zu können. Er kann sich die Beschreibung jedoch auch direkt vom Diensterbringer holen. Die Rolle des Vermittlers ist nicht nötig, wenn sich der Dienstnutzer und der Diensterbringer selbst über die Syntax und Semantik einigen können. Das setzt aber voraus, das sich beide vorher bereits kennen. Der Verzeichnisdienst in der Vermittlerrolle gehört jedoch neben der Rollen des Diensterbringers und des Dienstnutzers zu den zentralen Rollen des WS-Rollenmodels. Es ist nämlich in den meisten Fällen so, dass sich die Parteien zu dem Zeitpunkt wo sie miteinander kommunizieren wollen noch nicht kennen. In solchen Fällen wird solch ein Vermittler

benötigt um einen Webservice bekannt zu machen. Im folgenden wird gezeigt wie die gemeinsame Basis für die Nutzung von Webservices aufgebaut werden kann. Einige der möglichen Szenarien werden hier beschrieben.

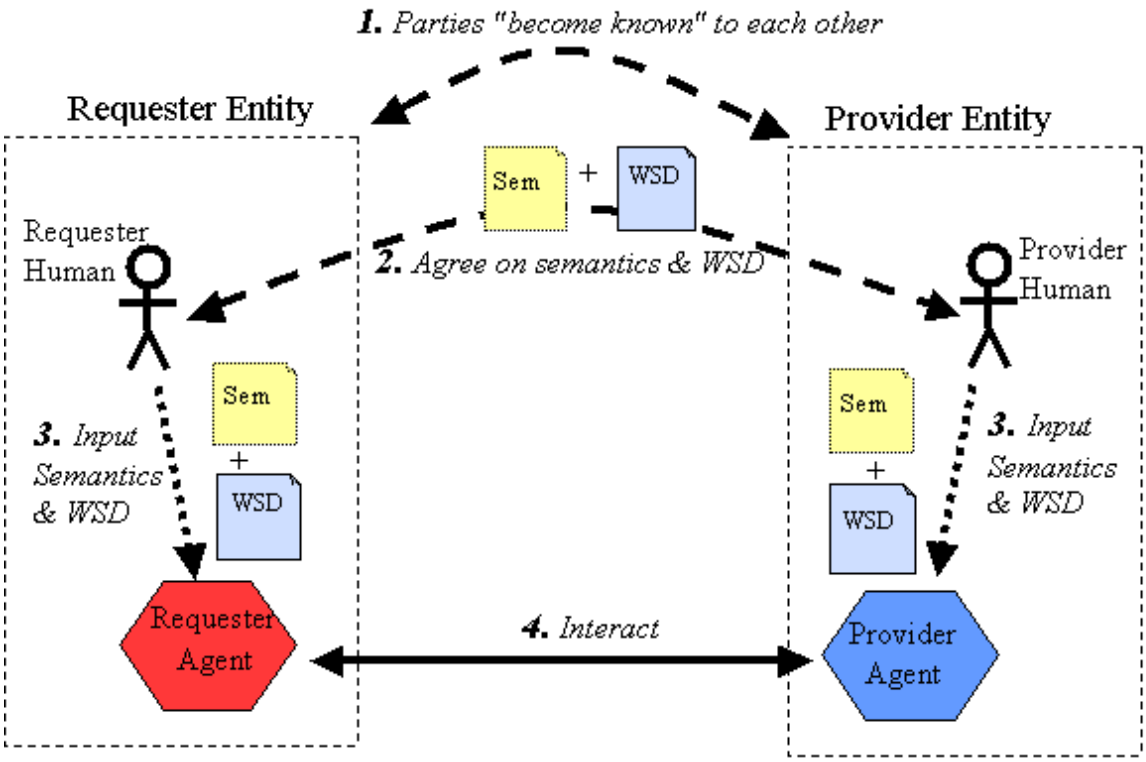

Abb. 2.2.2 Abbildung aus der W3C-Spezifikation. Szenario 1.

Das erste Szenario setzt voraus, dass sich der Dienstnutzer und der Diensterbringer vor der Interaktion bereits kennen. Sie einigen sich über die Semantik und die Syntax (d.h. die WSDL) und tauschen diese untereinander aus. Aufgrund dieser Informationen wird auf beiden Seiten die entsprechende Software erzeugt. Anschließend kann die Interaktion statt finden. Dieses Szenario wird dort zu finden sein, wo eine Zusammenarbeit von mindestens zwei Parteien bereits seit einiger Zeit bestanden hat und diese nun automatisiert werden soll.

In einem weiteren Szenario werden mehrere Diensterbringer berücksichtigt. Einige von denen bieten Services an welche, die gleiche Semantik erfüllen. Der Benutzer weiß was er erreichen möchte und kann sich den Vermittler zur Hilfe nehmen, um den geeigneten Service zu finden. Der Vermittler bzw. das Discovery-Tool kann entweder selbst aktiv nach neuen Services Ausschau halten oder lässt die Anbieter ihre Services selbst registrieren. Dabei müssen, der Webservice und die WSDL nicht unbedingt beim Verzeichnisdienst gespeichert sein. Es würde auch ausreichen, die entsprechenden Verweise dort zu pflegen.

Im nächsten Szenario erstellt der Diensterbringer nach der Einigung über die Semantik seine Software und publiziert beim Vermittler die WSDL mit der Semantik-ID. Der Nutzer programmiert seine Clientsoftware auf Basis der Semantikinformation. Die Software selbst sorgt zur Laufzeit dafür, dass mit Hilfe des Vermittlers der geeignete Service gefunden und eine geeignete WSDL geladen wird.

Die Semantik d.h. die Bedeutung eines Dienstes wird heutzutage als Dokument oder als Kommentar zusätzlich zur Dienstbeschreibung abgelegt. In Zukunft soll auch der Umgang mit der Semantik automatisiert ablaufen können. Dafür wird viel Forschung in diesem

Bereich unter dem Begriff des „Semantic Web“ betrieben. Die Basis dieser Idee bildet die formale Beschreibung der Semantik.

Das für die Arbeit relevante Basiswissen über Webservices, wurde in diesem Abschnitt behandelt. Um weiteres über Webservices zu erfahren verweise ich auf weiterführende Literaturquellen (s.Literaturverzeichnis). Nun sollen Webservices anderen Middlewaresystemen gegenübergestellt werden.

B. Webservices – Vergleich der Middlewaresysteme

Es sind bereits einige Middlewareimplementierungen entstanden, die an dieser Stelle näher betrachtet werden sollen. Dabei kann jedes von denen, einem der drei bekannten Ansätze der Softwareentwicklung zugeordnet werden. Es handelt sich dabei um den prozeduralen-, den objektorientierten- und den serviceorientierten Ansatz der Softwareentwicklung.

Der prozedurale Ansatz:

Die meisten RPC-Systeme unterstützen nur die synchrone Kommunikation zw. Client und Server, da die asynchrone Variante sehr komplex und schwierig zu implementieren ist. Abgesehen davon wird oft die peer-to-peer-Interaktion nicht unterstützt. Aus diesen Gründen sind solche Systeme nicht für Applikationen geeignet, die mit verteilten Objekten oder überhaupt nach dem OO-Prinzip arbeiten. Eine der Stärken der RPC-Systeme sind die synchronen Mechanismen zur Verhinderung der Netzüberlastung obwohl eine intensive Nutzung von Recoverymechanismen das jeweilige Netz trotzdem überlasten könnte. Aufgrund des Einsatzes statischer[4] Routingtabellen ist die Lastenverteilung (load balancing) schwierig zu realisieren. RPC-Systeme sind meistens proprietär, untereinander inkompatibel und oft abhängig vom Betriebssystem.

Nachrichtenorientierter Ansatz:

Dieser Ansatz wird durch so genannte Message-Oriented Middleware repräsentiert. Zu den Repräsentanten gehören vor allem Systeme wie MQ Series oder MS BizTalk. Der Kern dieses Ansatzes liegt in der asynchronen Kommunikation. Durch die Verwendung einer asynchronen Kommunikation zwischen der Client und der Serveranwendung, ist eine Kommunikation möglich, auch wenn die beteiligten nicht zur gleichen Zeit aktiv sind. Bedarf an solch einer Pufferfunktionalität besteht bei dem zu realisierenden Projekt nicht, deshalb wird dieser Ansatz nicht weiter verfolgt.

Objektorientierter Ansatz:

Common Object Request Broker Architecture (CORBA) ist eine objektorientierte Lösung von der OMG. Bei Java Remote Method Invocation (RMI) handelt es sich um entfernte Methodenaufrufe nach dem OO-Prinzip. Es ist für die Entwicklung von Systemen geeignet, die ausschließlich in Java programmiert werden. Für Entwickler solcher Systeme bzw. Anwendungen, wird es vom Vorteil sein, entfernte Aufrufe in der gewohnten Sprache schreiben zu können, die sie bisher für Desktopanwendungen verwendet haben. Bei einer Neuentwicklung wird man sich jedoch nicht gerne an eine Programmiersprache binden wollen. Dann gibt es auch noch das Distributed Component Object Model (DCOM) von Microsoft. Systeme wie CORBA oder DCOM benutzen einen Object Request Broker als Vermittler zu den aufzurufenden Diensten bzw. Objekten. Im Fall von DCOM repräsentiert die COM-Komponente den Broker. DCOMs Verhalten kann durch den Einsatz eines, ebenfalls von Microsoft entwickelten, Transaction Processing Monitor, nämlich den Microsoft Transaction Server [MTS], erheblich verbessert werden.

[4] Zur Übersetzungszeit festgelegt

Serviceorientierter Ansatz:
Webservices folgen dem serviceorientierten Ansatz. Dieser Ansatz wird am besten durch Systeme repräsentiert, die als eine serviceorientierte Architektur aufgebaut sind. In solchen Architekturen werden Webservices als lose gekoppelte Komponenten verwendet um sowohl fachspezifische als auch technische Dienste zu realisieren. Die Webservicetechnologie haben wir bereits im vorangegangenen Abschnitt ausführlicher betrachtet, da dies die bevorzugte Technologie für das Diplomarbeitsprojekt ist. Der Serviceorientierte Ansatz eignet sich, aufgrund der hohen Sprach- und Plattformunabhängigkeit, Interoperabilität, Skalierbarkeit(aufgrund der nahezu unbegrenzten Einsatzfähigkeit im Internet), am besten von allen bisher vorgestellten Ansätzen für die Erstellung des mobilen Bootsverleihsystems.

Durch die Verwendung von Webservices für das zu implementierende System, wird es sehr an eine serviceorientierte Architektur (SOA) angelehnt sein, weshalb im Folgenden die Grundzüge einer solchen Architektur beschrieben werden.

Die Service Oriented Architecture (SOA)
Das Konzept einer serviceorientierten Architektur sieht die Implementierung wieder verwendbarer, voneinander unabhängiger und lose gekoppelter Services vor. Die Services können unabhängig von zugrunde liegenden Implementierungen über Schnittstellen aufgerufen werden, deren Spezifikationen öffentlich und vertrauenswürdig sind. Unsere Architektur sieht vor, dass die Schnittstellen als WSDL veröffentlicht werden und durch die Registrierung vertrauenswürdig sind. Mit einer solchen Architektur wird, außer der Absicht ein verteiltes System zu schaffen, meistens das Ziel verfolgt die Wiederverwendbarkeit der bereits entwickelten Softwarekomponenten zu ermöglichen und die Flexibilität durch lose Kopplung zu erhöhen. Mit loser Kopplung ist vor allem die komponentenartige Bauweise der Services an sich gemeint. Einzelne unabhängige Komponenten können bei Bedarf leicht ausgetauscht werden. Das Gegenteil davon wäre ein monolithisches System, dass sich nur schwer bzw. gar nicht verändern lässt. Innerhalb einer SOA unterscheidet man zwischen zwei Arten von Services. Einmal sind es, die Enterprise Services. In einer Unternehmensarchitektur sind es diejenigen Services, welche die Geschäftsprozesse des jeweiligen Unternehmens unmittelbar unterstützen. Sie bilden die elementaren fachlichen Bausteine einer Anwendungslandschaft. Ein Beispiel für Enterprise Services aus dem Bootsverleihszenario, ist die Ausflugsplanung. Im Hintergrund, d.h. für den Benutzer transparent, verrichten andere Services ihre Arbeit, wie z.B. die Herstellung der Sicherheit. Man spricht dann von technischen Services. Dazu zählt unter anderem auch die Funktionalität des Verzeichnisdienstes, der die verschiedenen Dienstanbieter verwaltet.

Der zusammenfassende Vergleich:
Die RPC-Systeme oder Systeme wie CORBA, DCOM und EJBs sind relativ festkoppelnd. Sie werden in homogenen Netzen genutzt und setzen proprietäre Formate ein. RPC-Systeme sind außerdem untereinander oft inkompatibel und betriebssystemabhängig.
Die Idee das Internet als eine Sammlung von lose gekoppelten Diensten benutzen zu können wird erst durch Webservices realistisch. Webservices legen Schnittstellen nach außen, die sowohl von menschlichen Nutzern als auch von Maschinen effektiv genutzt werden können ([Middlewarebeschreibung von SEI]).

3. Mobile Geräte

3.1 Marktrecherche – „state of the art“

Nun haben wir unsere Anwendung einer Klasse mobiler Anwendungen zugeordnet.
In diesem Kapitel werden Technologien untersucht und verglichen, die eine Entwicklung und den Einsatz mobiler Anwendungen und insbesondere ortsgebundener mobiler Anwendungen erlauben.

Zu den wichtigsten Technologien bei der Entwicklung von Webservices im Bereich mobiler Geräte zählen insbesondere das J2ME und Microsofts .NET Compact Framework.
Wir werden uns deshalb zunächst diese beiden Alternativen anschauen.

3.1 A. Webservices auf mobilen Geräten

Als Erstes werden die möglichen Laufzeitumgebungen mobiler Geräte erwähnt. So gibt es das .NET Compact Framework von Microsoft und das Java 2 Micro Edition von Sun. Letzteres wird hier ausführlich behandelt.

.NET Compact Framework
Die Entwicklung von Webservices im .NET Compact Framework ist durch die Integration in Visual Studio sehr komfortabel. Microsoft hat die Vorteile der Webservice-Idee sehr früh erkannt und in die .NET Technologie eingebunden. Doch ist VS nicht kostenlos und läuft ausschließlich auf Windowsplattformen. Wegen unserer Anforderung der Unabhängigkeit von Plattform und Betriebssystem, ist dieses Framework zur Erfüllung unserer Zwecke nicht geeignet.

Java auf mobilen Geräten (aktuelle Standards und Implementierungen)
Unter anderem wird in diesem Abschnitt die Architektur von J2ME vorgestellt, die wichtigsten Konfigurationen und Profile dieser Plattform erläutert. Die möglichen Endgeräte des zu entwickelnden Systems unterscheiden sich signifikant im Bezug auf ihre Leistungsfähigkeit, Speicherkapazität, aber auch die Darstellungsmöglichkeiten weichen sehr von einander ab. Auf der einen Seite hat man zum Beispiel einen leistungsstarken PC mit einem 17 Zoll großen Monitor, und auf der anderen Seite ein Mobiltelefon mit einem 176x220 Pixel großen Display. Und dann gibt es auch noch die PDAs die oft leistungsstärker sind. An dieser Stelle möchte ich jedoch anmerken, dass Mobiltelefone auf Grund des riesigen Absatzmarktes rasant weiterentwickelt werden und die Leistungsfähigkeit der heutigen PDAs bald erreichen sollten. Die 80-er Reihe der Nokia Handys beispielsweise, die so genannten „communicator“, sind den heutigen PDAs an Leistungsfähigkeit nicht unterlegen. Für alle der drei soeben erwähnten Geräteklassen gibt es in der Javawelt, der Leistungsfähigkeit angepasste Implementierungen. Die Standardimplementierungen bzw. die Spezifikationen werden von der Firma Sun Microsystems unter unterschiedlichen Namen angeboten. Die J2EE[5] wird für die Umsetzung großer Unternehmensapplikationen bzw. Serverapplikationen eingesetzt, die J2SE[6] ist etwas kleiner geschnitten, doch ist der Kern der J2SE-Bibliotheken der gleiche wie bei der J2EE. Die J2SE d.h. die Standardausgabe der Javaumgebung ist für Entwickler „kleinerer“ Applikationen gedacht, die auf Leistungsstarken und ressourcenreichen Geräten, wie z.B. den heutigen PCs. Die beiden Java-Umgebungen sind seit langem bekannt und in vielen Büchern vertreten. An dieser Stelle soll jedoch etwas

[5] Java 2 Enterprise Edition
[6] Java 2 Standard Edition

genauer auf die dritte Spezifikation der Javawelt und zwar der mobilen Javawelt eingegangen werden. Die J2ME ist die Java 2 Plattform Micro Edition für mobile Geräte.

Die nächste Grafik (Abb. 3.1.1) soll die Abgrenzung der Pakete der J2ME (CLDC und MIDP) von denen der J2SE verdeutlichen.

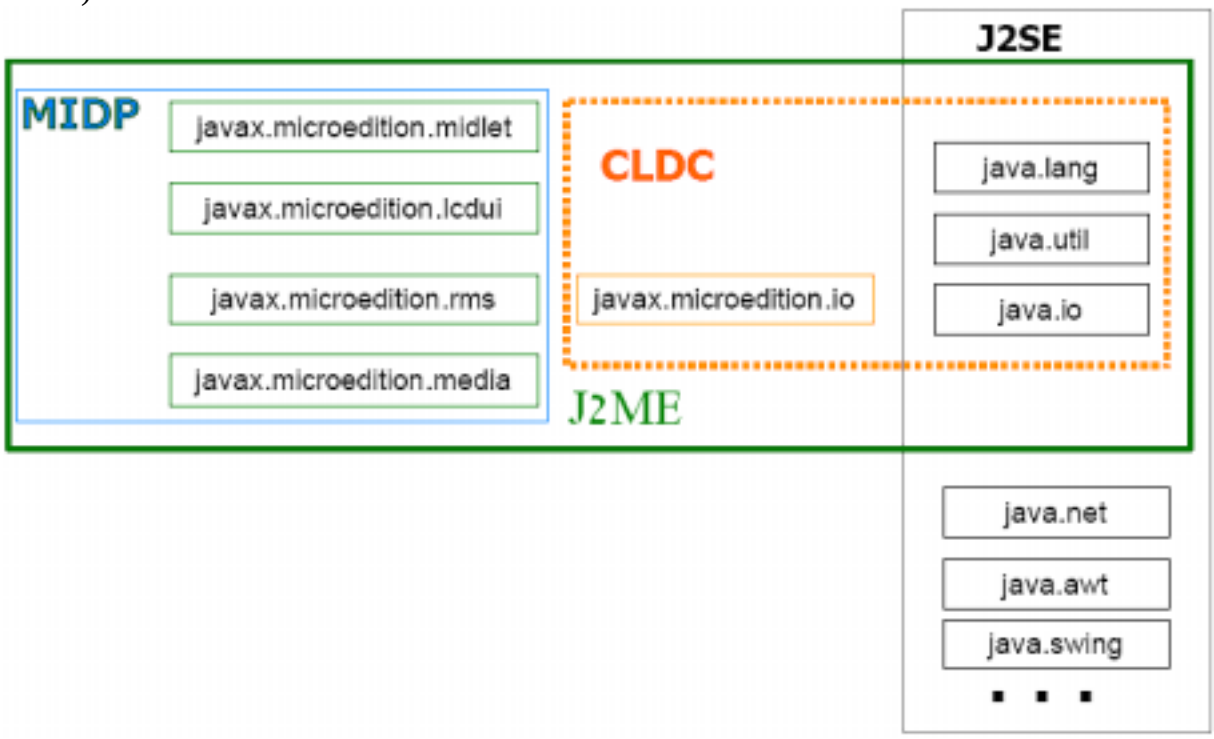

Abb. 3.1.1 Die J2ME Pakete der CLD-Configuration und des MID-Profils

Die J2ME Bibliotheken (s. Abb. 3.1.1), enthalten einige Pakete, die auch in der J2SE vorhanden sind, jedoch mit einigen Einschränkungen. Zu diesen Paketen gehören, das *java.lang*, das *java.util* und das *java.io*, welche die CLDC bilden. Die *javax.microedition.io* ist J2ME spezifisch. Die Pakete, java.net, java.awt und java.swing gehören ausschließlich zum Umfang von J2SE. Zu den Paketen der J2ME gehören auch noch die Pakete, javax.microedition.midlet, javax.microedition.lcdui, javax.rms und javax.microedition.media, welche das MIDP ausmachen. Auf CLDC und MIDP kommen wir bei der Erläuterung der nächsten Grafik zu sprechen.

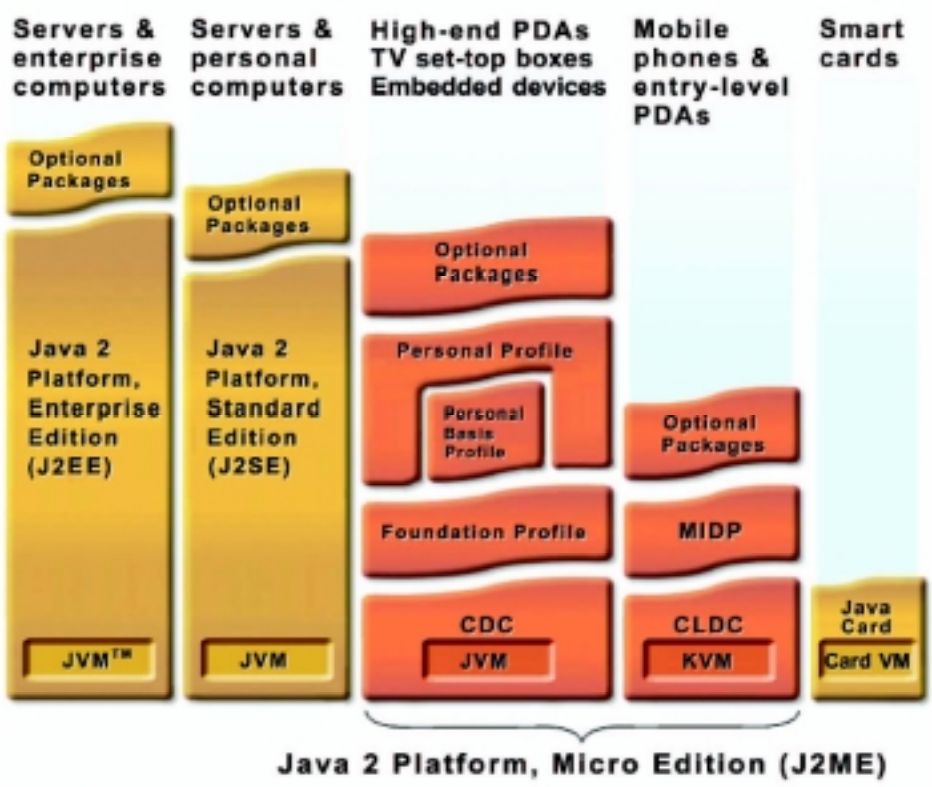

Abb. 3.1.2 - Javaplattformen

Allen Umgebungen gemeinsam, ist eine Virtuelle Maschine. Eine VM dient vor allem der Plattformunabhängigkeit und der Sicherheit, indem sie als ein Vermittler bzw. als eine Zwischenschicht zwischen dem Betriebssystem und der Programmiersprache fungiert. Zu der Geräteklasse der Standard Edition werden diejenigen Geräte zusammengefasst, die mehr als 10 Megabyte freien Speicher, eine hohe Prozessorleistung und eine komplexe Darstellung der Benutzeroberfläche bieten. Dazu zählen die Desktop Computer und die Laptops. Eine andere Geräteklasse bilden Geräte wie die high-end PDAs (s.Glossar unter PDA), die einen Speicher von 1 bis zu 10 Megabyte, eine geringere Prozessorleistung als z.B. die PCs und aufgrund ihrer kleineren Displays eine abgeschwächte Form der Oberflächengestaltung aufweisen. Für den Aufbau einer geeigneten und an die knappen Ressourcen angepassten Umgebung, wurden so genannte Konfigurationen und Profile entwickelt. Konfigurationen setzen dabei auf die virtuelle Maschine auf und die Profile auf die Konfiguration. Für die high-end PDAs wird für die Programmierung für gewöhnlich die J2ME in Kombination mit CDC[7] und dem Personal Profile verwendet. Falls eine Anwendung die verfügbaren Ressourcen des high-end PDAs nicht verwenden muss und sie unter 32kByte Größe bleibt, so kann auch das CLDC[8] mit MIDP[9] eingesetzt werden um beispielsweise bessere Portabilität zu erreichen. Denn so wird eine derartige Applikation auf Geräten aller Klassen ablaufen können sobald J2ME unterstützt wird.
Die Geräte der dritten Geräteklasse sind nämlich Mobiltelefone und die low-end PDAs. Für die Programmierung greift man eben auf die CLDC und MIDP Kombination der J2ME Umgebung. Diese Geräte verfügen über einen Speicher von 32 bis 512 kByte Größe, sehr geringe Prozessorleistung und eingeschränkte Fähigkeiten, die Oberfläche einer Anwendung darzustellen. Für diese Diplomarbeit erscheint das J2ME in der zu letzt beschriebenen Ausführung die geeignete zu sein insbesondere wegen der hohen Portabilität. Die Fähigkeiten

[7] Connected Device Configuration
[8] Connected, Limited Device Configuration
[9] Mobile Information Device Profile

der Leistungsstarken Geräte können dadurch natürlich nicht voll ausgenutzt werden, die geforderte Funktionalität ist jedoch realisierbar.

Die Unterste Schicht der J2ME Umgebung (wie in der Abb. 3.1.2 zu sehen) bildet die Java Virtuelle Maschine in Kombination mit einer so genannten Konfiguration. Für die Leistungsschwächste Geräteklasse musste eine beschnittene Version der Java Virtuellen Maschine entwickelt werden. So ist die Kilobyte Virtuelle Maschine (KVM) entstanden.
Die Konfigurationen sind Java-Bibliotheken, die Javabasisfunktionen und den Zugriff auf die virtuelle Maschine umfassen. Die KVM erfordert mind. 192 kB an verfügbarem Speicherplatz und die JVM satte 4,5 MB.

In der nächst höheren Schicht sind die Profile untergebracht. Es sind weitere Programmierbibliotheken die, die Konfigurationen wiederum um Gerätespezifische Funktionen, wie die Netzwerkkommunikation und Benutzungsschnittstellen, ergänzen. So wie das MIDP für Mobiltelefone und PDAs geeignet ist, so ist ein anderes Profil wie z.B. das Information Modul Profile (IMP) für eine Machine to Machine Kommunikation gedacht. Das MIDP ist für uns also das relevante Profil.Java-Applikationen, die auf Grundlage der MIDP entwickelt werden, nennt man MIDlets.

In MIDP wird ein Applikations-Modell eingeführt, das wohl am ehesten mit dem Applet-Modell zu vergleichen ist. Analog zu Applets gibt es unter MIDP so genannte MIDlets, die durch ein vordefiniertes API auf den mobilen Endgeräten ablaufen und dort eingeschränkte Rechte besitzen. Neben der eigentlichen Programmierung der MIDlets gibt es noch bei der Auslieferung einige Besonderheiten. Die verifizierten Java-Klassen eines oder mehrerer MIDlets werden zur Auslieferung in einem JAR-File zusammengefügt, welches genau dem JAR-Standard von J2SE entspricht. Weiterhin kann optional ein so genannter Java Application Descriptor (JAD) mitgeliefert werden, der Meta-Informationen über die MIDlets enthält. Dies ist sehr wichtig, da ja die Ressourcen auf den mobilen Geräten sehr eingeschränkt sind und ein Software-Download oft auch recht teuer ist. Vor dem Download wird zuerst das JAD-File interpretiert, damit der Endbenutzer die wesentlichen Daten wie Name, Größe, etc. erhält. Dieser kann dann entscheiden, ob er das eigentliche JAR-File herunterladen will.
Optionale Pakete bilden die oberste Schicht der J2ME Architektur und erweitern entsprechend die Kombination aus Konfiguration und Profil. Im Bezug auf diese Diplomarbeit kann erwähnt werden, dass die Spezifikation von Webservices in der J2ME das JSR172 als optionales Paket vorsieht. Für diese Arbeit ist es nun wichtig die geeignete Implementierung der gerade vorgestellten Programmierumgebung zu finden. Zu Beginn des neuen Jahrtausend gab es relativ viele Javaimplementierungen für kleine Geräte die sich bis heute noch nicht oder gar nicht durchgesetzt haben und viele sind sogar ganz von der Bildfläche verschwunden. Von ihrer früheren Existenz zeugen nur noch einige „tote“ Links im World Wide Web. Mögliche Ursachen für das Verschwinden war oft eine komplizierte Installation und fehlende Dokumentation. Dafür gibt es heute einige komfortable Lösungen, die oft durch geeignete Entwicklungswerkzeuge abgerundet werden. Einige Handyhersteller wie z.B. Nokia haben Java fest in ihre Marktstrategie eingebunden und entwickelten eigene Lösungen die den Installationsaufwand enorm reduziert haben. Der Programmierer kann sich somit mit seiner eigentlichen Aufgabe beschäftigen, nämlich dem Programmieren. Auch IBM bringt mit dem IBM WebSphere ein Produkt auf den Markt, welches sich für den Einsatz auf dem mobilen Client unseres Systems gut eignen würde. Doch da es nicht kostenlos ist und die Nutzung der Testversion eine zu kurze Nutzungsdauer hat, wird dieses Produkt nicht verwendet.

IBM WebSphere
Das *IBM WebSphere Everyplace Micro Environment*: Diese Komponente des IBM-Produkts "*Workplace Client Technology Micro Edition*" unterstützt Java-Laufzeitumgebung der Java 2 Micro Edition (J2ME). Die Laufzeitumgebung gibt es in den folgenden Varianten:
a) WebSphere Everyplace Micro Environment MIDP 2.0 für Windows Mobile 2003
b) WebSphere Everyplace Micro Environment Personal Profile 1.0 für Windows Mobile 2003.
c) WebSphere Everyplace Micro Environment MIDP 2.0 für Palm OS
d) WebSphere Everyplace Micro Environment Personal Profile 1.0 für Sharp Zaurus

Die offiziellen Angaben zum Support auf der IBM-Homepage [JavIBM]*:*
Connected, Limited Device Configuration (CLDC 1.0 and 1.1) and Mobile Information Device Profile (MIDP 2.0) for the palmOne Tungsten C and HP iPAQ 4700, HP iPAQ 3715, and HP iPAQ 5550 devices.
Connected Device Configuration (CDC 1.0_01), Foundation Profile, and Personal Profile for the HP iPAQ Pocket PC h5550 and the Sharp Zaurus SL-6000
Hier sind die Geräte in ihrem Auslieferungszustand gemeint d.h., dass die Informationen sich auf IPAQs beziehen, die mit dem Betriebssystem Windows CE ausgestattet sind.

Aufbauend auf diese Laufzeitumgebungen, bilden die folgenden Technologien, die von Grund auf für mobile Geräte konzipiert wurden, eine Basis für weitere Entwicklung von Webserviceapplikationen auf derartigen Geräten.

Mobile Webservices im Javaumfeld
Wie bereits erläutert wurde, sind für den Zugriff auf das zu entwickelnde System mobile Geräte vorgesehen, jedoch gibt es viele verschiedene Ausführungen mobiler Geräte im Umlauf. So sollte das System Technologien nutzen, die auf möglichst vielen Geräten laufen können oder besser noch bereits vorinstalliert sind. Dafür wird zumindest eine Laufzeitumgebung benötigt, die auf relativ vielen Plattformen unterstützt wird. Die Antwort darauf könnte die Internetprogrammiersprache Java sein, deren Laufzeitumgebung für kleine mobile Geräte als J2ME bekannt ist. Java wird derzeit auf sehr vielen mobilen Plattformen unterstützt. Dafür bedarf es einer Java Virtuellen Maschine und entsprechender Programmierbibliotheken.

Für PCs gibt es für XML das Xerces-package, das Axis für SOAP und einige weitere Implementierungen. Diese sind jedoch zu groß und verbrauchen zu viele Ressourcen als, dass sie für mobile Geräte wie die PDAs oder Mobiltelefone einsetzbar wären.

Hier sind die wichtigsten Lösungen für Webservices in Java auf mobilen Geräten:

- JSR172 Spezifikation von Sun. - Umfasst XML, SOAP und Webservices für mobile Geräte.
- kSOAP von Enhydra.org

Beide Lösungen werden im Folgenden grob vorgestellt. Weiteres über Webservices im Kapitel 3.2 - Webservices – Vorteile im Vergleich zu anderen Verteilten Systemen
.
J2ME – Spezifikation zu Webservices – das JSR 172
Erstellt im Rahmen des JCP (Java Community Process) in 2003. Das Letzte Release wurde am 03. März 2004 erstellt. Die Spezifikation ist die Antwort auf ein besonderes Interesse der Entwicklergemeinde im Bezug auf die Erweiterung der J2ME-Clients um die Enterprise

services. Diese sollen die fachlichen Services einer Service Oriented Architecture (SOA) abbilden..
Zum Parsen der XML-Daten wurde ein kleiner Teil der JAXP 1.2 Funktionalität gewählt. Für den Zugriff auf Webservices hat man einen Teil der JAX-RPC 1.1 Funktionalität selektiert. Auf die Serverfähigkeit musste verzichtet werden. Außerdem wurde auf DOM und XSLT Funktionalität verzichtet. Der Grund für die Sparmassnahmen sind die Knappen Ressourcen der mobilen Geräte. Der Verzicht auf DOM, das zur Manipulation von XML-Dokumenten gebraucht wird, ist nicht besonders tragisch, da dessen Funktionalität mit einem SAX-Parser (s. Glossar –SAX) mit etwas mehr Programmieraufwand erreicht werden kann. Was das eben angesprochene XSLT betrifft, so kann es eher bei serverseitigen als bei clientseitigen Anwendungen ihren Einsatz finden, d.h. auch dieser Verlust fügt unserer Anwendung keinen Schaden zu, weil die mobilen Geräte nicht als Server, sondern als schlanke Zugriffsclients verwendet werden sollen, wie es heutzutage noch üblich ist.

kSOAP
Das kSoap von Enhydra.org ist eine Java-Implementierung für SOAP. Die API bietet Funktionen zur Erstellung von SOAP-Anfragen. Durch die integrierte Implementierung von kXml können SOAP-Antworten geparst und ausgewertet werden.
Das kSOAP basiert auf dem Konzept der dynamischen Aufrufe (s.DII im Glossar). Stubklassen werden nicht generiert. Durch "dynamic typing" kann die Typüberprüfung nicht vom Compiler durchgeführt werden, was theoretisch ein Nachteil ist, hinsichtlich der Performance und der Fehlersuche. Mit der so gewonnenen Flexibilität wird nämlich einiges an Arbeit erst zur Laufzeit erledigt und kann nicht mehr vom Compiler übernommen werden. So wird die Performance durch den zusätzlichen Aufwand der Typprüfung zur Laufzeit negativ beeinflusst. Die Fehlersuche des Compilers kann somit die Typüberprüfung nicht mit einbeziehen.
Das kSOAP bietet eine Möglichkeit eigene Datenstrukturen zu serialisieren. Das entsprechende Interface heißt *KvmSerializable*. In der neuen Version von kSOAP, der kSOAP2, die für den mobilen Client dieser Arbeit verwendet wird, ist diese Schnittstelle nicht mehr sichtbar. Dafür werden so genannte Marshallerklassen vom Entwickler direkt bearbeitet. Alle Serialisiererklassen werden nicht mehr bei der *ClassMap* registriert, sondern bei der *Envelope* registriert werden. Damit ist die Envelope das neue Interface von kSOAP.

Vorgehensweise bei der Nutzung von kSOAP:

- Eigene Datenobjekte werden mithilfe von Marshallerklassen serialisiert.
- Datenobjekte werden, sowohl auf der Server wie auch auf der Clientseite, mit der Envelope registriert.
- Integration eigener Services in KSoapServlet. (Optional kann auch ein eigenes Servlet kreiert werden, falls besondere Funktionalität gewünscht wird)

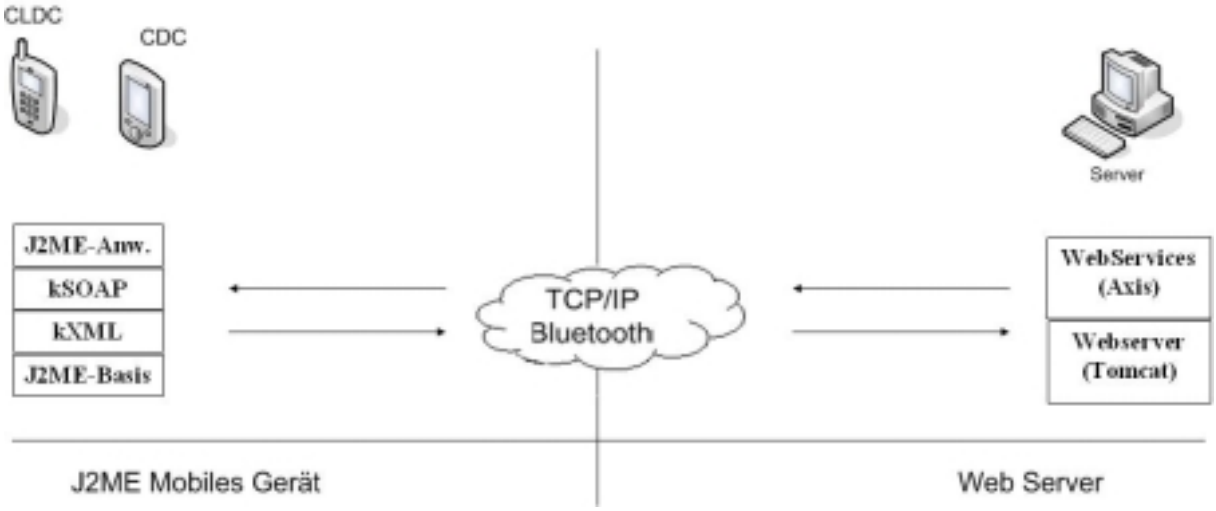

Abb. 3.1.3 – kSOAP-client und Axis-server

Die Abbildung 1 zeigt, wie die Architektur eines mobilen Clients unter Verwendung von kSOAP aussieht. Der Webservice kann, muss jedoch nicht zwingend, ein Axis-Webservice sein.

Serverseitige Lösungen für Webservices
Da die Entscheidung, Java als Programmiersprache zu verwenden bereits gefallen ist, wird es sinnvoll sein (um den Einarbeitungsaufwand zu minimieren) auch Serverseitig in diese Richtung zu gehen. Hier gibt es das Apache Axis. Wir werden die Version 1.2.1 verwenden. Als Alternative zu Axis kann z.B. Apache Soap benutzt werden. Da für den Betrieb des Prototypen ein Windowsrechner eingesetzt wird, wäre auch die Verwendung von .NET möglich, doch wollen wir uns nicht auf eine Plattform festlegen. Es sehr wahrscheinlich, dass einige Server von Bootsverleihern andere Betriebssysteme als Windows verwenden.

Alternative Lösungen für Webservices
Man kann Webservices auch mit der Skriptsprache Python entwickeln, mithilfe der SOAP - Bibliothek *ZSI* (Zolera SOAP Infrastructure) oder *SOAPpy*. Python ist eine interpretierte, interaktive und objektorientierte Skriptsprache. Sie kann auf Unix bzw. Linux und Windows Plattformen eingesetzt werden. Python gilt als leicht erlernbar und erfreut sich deshalb hoher Beliebtheit bei den Programmierern. Um einen Python-Webservice anbieten zu können, muss der Server die Kommunikation über http und das XML-Parsing unterstützen. Außerdem ist der Apacheserver mit PHP4 (bzw. neuere Version) notwendig und natürlich auch eine Pythonimplementierung.

Mobile Webservices – Bewertung und Blick in die Zukunft
Der Zuwachs an mobilen Geräten wird wohl noch lange anhalten. Damit sich aber Webservices, im Bereich mobiler Geräte, erfolgreich durchsetzen können, müssen einige Probleme gelöst werden. Gemeint sind hier Probleme wie: Energieeffizienz, server capability specification, disconnected computing, Unterstützung für peer-to-peer-Netzwerke, und die s.g. quality of service. Doch gibt es bereits jetzt gute Möglichkeiten, mobile Anwendungen auf Webservicebasis zu entwickeln. Das .NET Compact Framework ist sehr gut auf Webservices abgestimmt und die Portabilität erhöht sich dadurch, dass sehr viele der mobilen Geräte, die zur Zeit auf den Markt kommen, mit diesem Framework ausgestattet sind, doch die Plattformabhängigkeit und der Preis, belasten immer noch das Image des Frameworks. J2ME ist immer noch auf den meisten Geräten vorhanden und hat z.B., neben der hohen Portabilität, einen finanziellen Vorteil. Die Umgebung kann nämlich kostenlos aus dem

Internet herunter geladen werden, so dass auf Geräten wo diese nicht vorinstalliert sein sollte, eine Neuinstallation kein Problem darstellt. Zumindest kein finanzielles. Auf J2ME aufbauend kann das kSOAP-Framework für die clientseitige Webservicefunktionalität sorgen, wobei auch dieses Werkzeug kostenlos ist.
Auf die Entwicklung mobiler Webservices hatten insbesondere die beschränkten Ressourcen, das ist der geringer Speicher- und Energie – Verbrauch, großen Einfluss. In diesem Zusammenhang sollte man jedoch erwähnen, dass wenig Speicherverbrauch nicht unbedingt weniger Energieverbrauch bedeutet. Oft ist es sogar genau umgekehrt. Ein Webbrowser ohne einen Cachingmechanismus verbraucht sicherlich weniger Speicher als einer mit Caching, es wird jedoch auch mehr Energie kosten die mehrmals aufgerufenen Seiten immer wieder neu zu laden.

Kapitel 4. Analyse

Das Analysekapitel soll dem Leser einen Überblick über die Funktionalität des zu entwickelnden Systems verschaffen und die Interaktion der Akteure mit dem System schildern. Als Akteure werden z.B. Benutzer der mobilen Geräte und die Bootsverleiher bezeichnet, die das System benutzen.
Damit deutlich wird welche Funktionalität das System erfüllen muss, werden im Kap. 4.1 die Anforderungen spezifiziert. Die Interaktion zwischen den Akteuren und dem System wird im Kap. 4.2 analysiert. Kapitel 4.3 befasst sich mit der Funktionalität des Systems, die mithilfe fachlicher Komponenten beschrieben wird. Der Begriff fachliche Komponenten als auch der Begriff funktionale Komponenten wird in dieser Arbeit synonym behandelt. Durch diese Komponenten wird die fachliche Funktionalität beschrieben. Am Ende des Analysekapitels (im Kap. 4.4) erfährt der Leser, welche Systemarchitektur für das System geeignet ist und welche letztendlich für die Umsetzung verwendet wurde.

Das zu entwickelnde System wird „Mobiler Ortsbasierter Bootsverleih“ heißen. Ortsgebundene Angebote von Dienstleistern sollen auf mobilen Geräten auffindbar und buchbar sein. Bei den Angeboten handelt es sich um Boote und bei Dienstleistern um die Bootsverleiher, welche diese anbieten. Die Angebote sollen sich möglichst in der Nähe des Aufenthaltsortes befinden. Der Weg zum Bootsverleiher soll als Route angegeben werden.
An das System werden Anforderungen gestellt, die im folgenden Kapitel (Kap. 4.1) formuliert werden.

4.1 Analyse - Abgeleitete Anforderungen

Folgende Anforderungen werden an das zu entwickelnde System gestellt:

Ortsgebundene Angebote
Verstreute Dienste von Bootsverleihern, müssen beim Verzeichnisdienst ortsgebunden registriert (publiziert) werden können. Angaben über die Lage (ortsgebunden) des Verleihs werden bei der Registrierung des Dienstes angegeben.
Das Suchen nach registrierten Leihstellen und nach Booten in einer bestimmten Umgebung soll ermöglicht werden. Unter Verwendung eines GIS können wir mehrere Modelle für eine Suche nach Booten verwenden. Zum einen ist es die direkte Suche nach Booten, zum zweiten kann zuerst nach Dienstleistungen gesucht werden. Diese wiederum ermöglichen die Suche nach Booten. Zum Dritten, kann eine Umgebungssuche gestartet werden, welche die Dienstleistungen einer bestimmten Umgebung angibt. Eine Routenberechnung soll angeboten werden.

Benutzernavigation
Auf Wunsch des Nutzers sollen Landkarten angezeigt werden können. Mögliche Einsatzmöglichkeiten für eine Kartenausgabe sind die Ergebnisse der Umgebungssuche und der Routenberechnung.

Konventionen
Damit die Dienste der Bootsprovider von potenziellen Kunden in Anspruch genommen werden können, müssen sich die Dienstanbieter an einige Konventionen (Vereinbarungen) halten bzw. müssen Vereinbarungen zwischen der Basis- und der Providersoftware getroffen werden. Mit Basissoftware sind, die Clientsoftware und der Verzeichnisdienst gemeint.
Solche Vereinbarungen sind unvermeidbar, solange sich der Datenbestand bei den Providern selbst befindet. Denn, ob lediglich der Dienst oder die Boote selbst publiziert werden, es muss

in jedem Fall eine Schnittstelle für den entfernten Datenzugriff auf die Providerdienste geben. Mit Datenbestand sind hier insbesondere die Bootdatensätze gemeint.

Konventionen, an die sich die Providersoftware halten muss, sind folgende:
Das erste was der Provider tun muss, um am Verleihsystem teilnehmen zu dürfen ist, dass er einen eigenen Dienst Implementiert und diesen beim Verzeichnisdienst registrieren lässt (Der Begriff der Registrierung wird im Text synonym zum Begriff des Publizierens verwendet). Dieser Dienst muss Funktionen fürs Suchen nach Booten und das Reservieren von Booten beinhalten. Außerdem muss der Provider, im Fall einer Dienstpublizierung, den Verzeichnisdienst kontaktieren.

Schnittstellenintegration:
Sowohl der Verzeichnisdienst als auch die Clientsoftware des zu entwickelnden Systems, werden Schnittstellen zur Verfügung stellen, gegen welche die Provider ihre Software programmieren können.
Die Dienste der Bootsverleiher dagegen stellen externe Funktionalitäten dar, die es in das System bei Bedarf einzubinden gilt. Diese Dienste sollen einheitlich angesprochen werden können, somit entsteht das Problem der Einigung über gemeinsame Schnittstellen.
Die Integration neuer Funktionalitäten bei komplexen Legacysystemen (bestehende, meistens monolithische Systeme) bedarf großer Anstrengung bzw. ist in manchen Fällen nahezu unmöglich. Abhilfe bei Systemen, die noch entwickelt werden sollen, können Organisationen wie die OASIS schaffen, die für die Entwicklung von Standardisierungen im Bereich des E-Business zuständig sind. Die Hersteller von Softwaresystemen berücksichtigen diese Standards, da diese die Akzeptanz beim Kunden für das jeweilige System steigert. Als großer Nachteil ist jedoch die lange Etablierungszeit solcher Standards zu nennen, denn diese Systeme müssen erst entwickelt werden. Solche monolithischen Systeme gehören zu der veralteten Softwaregeneration. Heute, in Zeiten der Globalisierung, wo das Internet den Weltmarkt für Alle zugänglich macht, wird eher zu Serviceorientierten Lösungen gegriffen. Die OASIS entwickelt unter anderem Standards, die eine reibungslose Integration von Funktionalitäten in bestehende Geschäftsabläufe unterstützen sollen – Stichwort „Supply Chain“. Standards dieser Art werden bei relativ großen E-Business-Lösungen ihren Einsatz finden.
Für das relativ kleine Bootsverleihsystem, das von Grund auf neu und als eine serviceorientierte Lösung entwickelt werden soll, werden Schnittstellen definiert, an die sich die teilnehmenden Entwickler der Providersoftware halten müssen. Das Problem gemeinsamer Schnittstellen wird auf diese Weise gelöst.

Gemeinsame Beschreibungsmerkmale:
Da sich, wie bereits erwähnt, die Boote bei den Bootsverleihern befinden, wird die Suche nach Booten bei den Verleihern selbst stattfinden. Somit muss sowohl die Clientsoftware als auch die Providersoftware die gleichen Ausleiheigenschaften bei der Suche berücksichtigen.

Als Suchkriterien können die Produkteigenschaften, Dienstleistungseigenschaften und ortsbezogene Eigenschaften genutzt werden.

Folgende Produkteigenschaften werden für die Suche verwendet.
Zeitraum der Ausleihe, Kapazität an Personen, die Bootsart, die Art des Antriebs und die Leistung des Motors (falls vorhanden).
Durch die Angabe eines Ausleihzeitraums kann die Suche die Boote ausschließen, die in der gewünschten Zeit bereits anderweitig verliehen werden.

Abgesehen von der Zeitraumangabe, betreffen die genannten Angaben, die Eigenschaften von Booten. Auch diese Angaben werden für eine gezielte Einschränkung der Suchergebnisse sorgen.

Dienstleistungseigenschaften dienen als Suchkriterien auf einer höheren Ebene.
Die *Antriebsart* wird bei dem Prototypen diese besondere Aufgabe erfüllen.
Die *Antriebsart* soll einerseits vom Bootsverleiher bei der Bootsuche berücksichtigt werden. Das passiert auch bei den anderen Angaben, die das Boot beschreiben. Andererseits soll sie die Suche bereits auf der Ebene der Dienstleistungssuche beim Verzeichnisdienst einschränken. Es handelt sich dabei um eine wichtige Funktionalität eines Verzeichnisdienstes, weshalb sie hier eingebracht wurde. Auch andere Attribute könnten auf der Ebene abgeprüft werden. Vorausgesetzt, der Verzeichnisdienst und das jeweilige Providerobjekt sieht dieses vor. D.h. diese müssten um das Attribut erweitert werden.

Eine weitere nützliche Sucheinschränkung sollen die *ortsbezogenen Eigenschaften* bilden. Damit sind die Angaben für die Suchumgebung gemeint. Dabei handelt es sich um eine beliebige Adresse in der Welt und den Radius zur Bestimmung der Suchumgebung.

4.2 Analyse - Anwendungsfälle

Anhand von Anwendungsfällen wird hier versucht, sowohl grafisch als auch textlich darzustellen wie das System mit seinen Akteuren interagiert. Eine Folge von Interaktionen führt zu einem, für den Akteur nützlichen, Ergebnis.
Die eingeführten Anwendungsfälle werden im Kap.5 Design detailliert ausgearbeitet. Die fachliche Funktionalität des Systems wird weiter unten in diesem Kapitel unter Analyse-fachliche Systemkomponenten beschrieben.

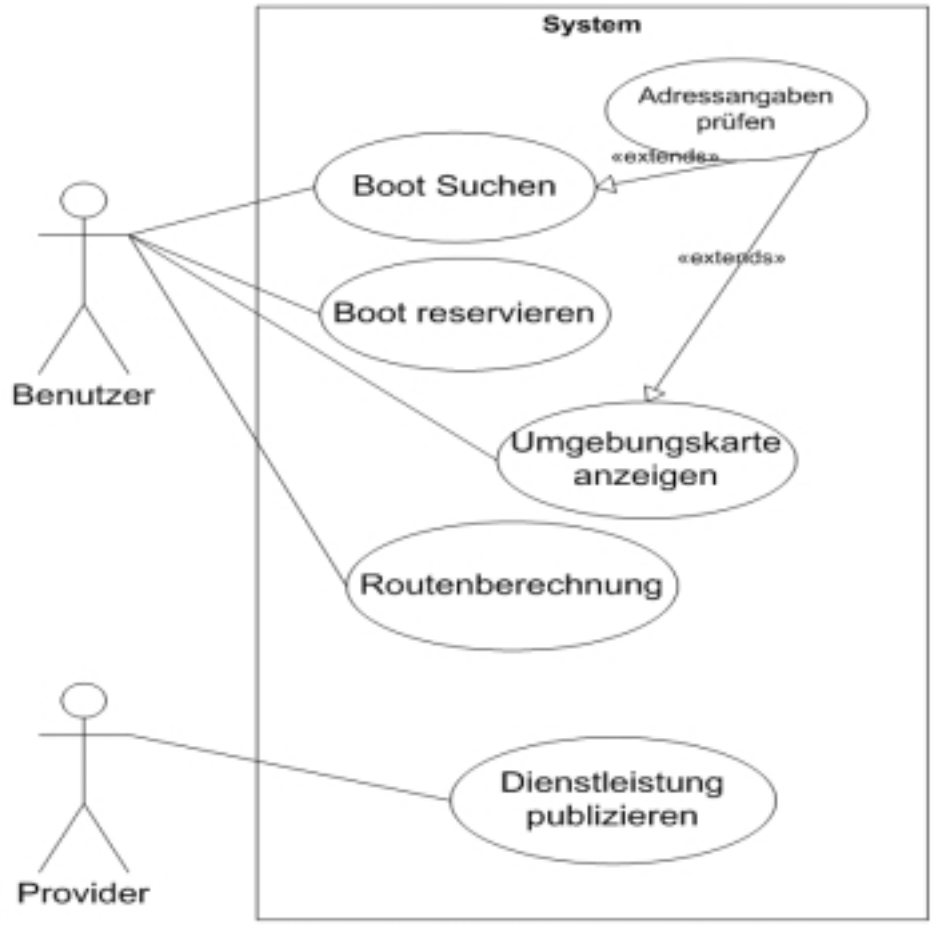

Abb. 4.2.1 Anwendungsfalldiagramm

Erfassen und Abschicken von Benutzerangaben
Bevor überhaupt etwas vom System angefragt werden kann, muss der Benutzer einige Angaben zur Ausleihe machen.

Die folgenden beiden Grafiken zeigen jeweils ein Formular der mobilen Anwendung. Im ersten Formular werden die so eben besprochenen Ausleiheigenschaften erfasst. Im zweiten die Angaben, für die ebenso bereits genannte Umgebungssuche. Für Handynutzer müssen an dieser Stelle Auswahlmenüs angeboten werden. Dadurch kann der Nutzer z.B. die Barkasse als Bootsart auswählen und muss es nicht mühsam mit den kleinen Handytasten eintippen. Aus dem gleichen Grund werden die Boote in Leistungsklassen unterteilt.

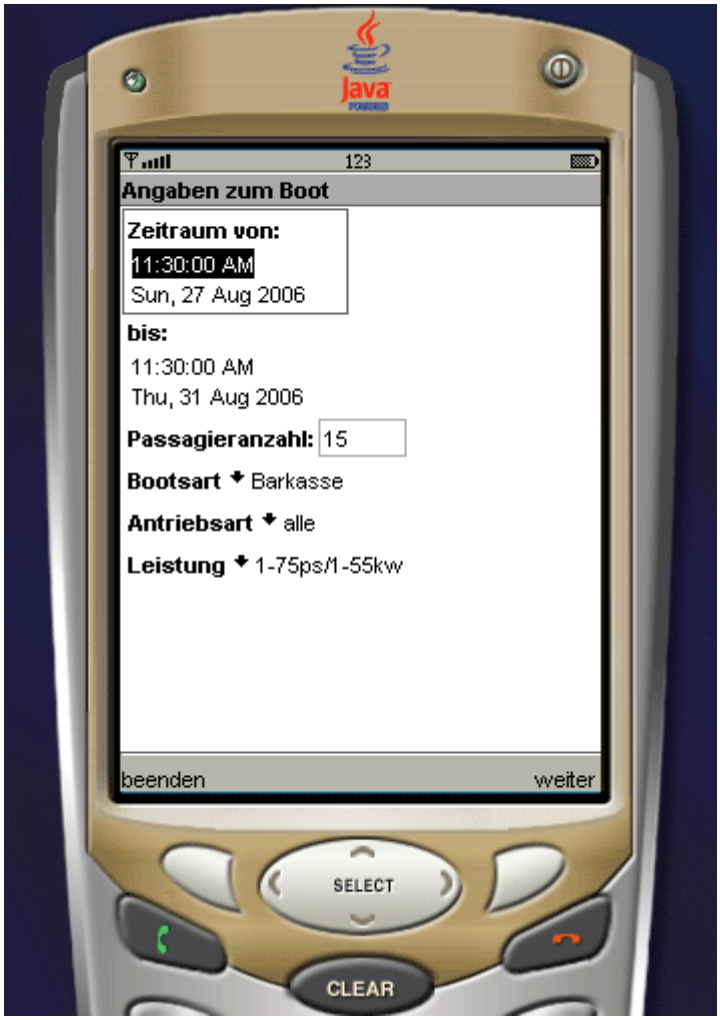

Abb. 4.2.2 Formular der mobilen Anwendung – Angaben zu den Ausleiheigenschaften

Die Angaben der Form, die in der obigen Grafik gezeigt werden, müssen an die jeweiligen Bootsprovider weitergereicht werden, damit diese die endgültige Angebotsliste erstellen können. Die einzelnen Angaben wurden im Vorfeld dieses Kapitels bei der Beschreibung von Konventionen behandelt.

Das Formular mit möglichen Eingaben für die Umgebungssuche wird in der folgenden Grafik gezeigt.

Abb. 4.2.3 Formular der mobilen Anwendung – Angaben für die Umgebungssuche

Die Angaben, die innerhalb dieser Form gemacht werden, betreffen den Ausgangspunkt und den Suchradius der Umgebungssuche. Die Komponente *Umgebungssuche* wird sowohl von der Suchkomponente *Bootsuche* als auch von der Komponente *Umgebungskarte erstellen* verwendet. Die Komponenten werden weiter unten im Abschnitt 4.3 erläutert.

A. Anwendungsfall – Adressangaben Prüfen

Sobald die Benutzerangaben abgeschickt wurden, wird als erstes eine Prüfung der Adresse des Zentrums der Umgebungssuche durchgeführt.

Hauptszenario: Adressangaben prüfen

- Der Benutzer macht Angaben über die Ausleihe und bestätigt,
- macht Angaben zur Umgebungssuche und schickt die Angaben ab,
- wartet auf Antwort,
- erhält ein Menü zu sehen, wo er sich für eine Anzeige der Umgebungskarte oder für die Bootsuche entscheiden kann. (Beides wird als einzelne Anwendungsfälle weiter unten beschrieben)

Für den Fall, dass eine Adressangabe nicht eindeutig ist, werden alle potenziellen Kandidaten zur Auswahl angeboten. Solch ein Fall kann z.B. auftreten, wenn der Benutzer keine Postleitzahl angegeben hat und die angegebene Strasse über mehrere Postleitzahlenbereiche gespannt ist. Das ist z.B. in Hamburg bei der Stresemannstrasse der Fall (s. Abb. 4.2.5).
Solche Vorfälle sind eher die Ausnahme, weshalb ein solches Szenario als ein Ausnahmeszenario behandelt und im Folgenden beschrieben wird.

Ausnahmeszenario: Adressenkonflikt

- Der Benutzer macht Angaben über die Ausleihe und bestätigt,
- macht Angaben zur Umgebungssuche und schickt die Angaben ab,
- wartet auf Antwort,
- wegen einer nicht eindeutigen Adressangabe, erhält er eine Liste mit Adressvorschlägen zur Auswahl.

Abb. 4.2.4 Ermitteln der Adressen bei nicht eindeutigen Adressangaben

Als erstes nach dem Abschicken der Benutzerangaben, werden die Adressangaben auf Eindeutigkeit überprüft. Das nimmt ein wenig Zeit in Anspruch, so muss ein entsprechender Wartescreen (s. Abb. 4.2.4) die Bearbeitung ankündigen.

Abb. 4.2.5 Liste der Adressvorschläge für den Ausgangspunkt

Um den Adresskonflikt vorführen zu können wird statt Seehafenstrasse (s. Abb. 4.2.3) Stresemannstrasse (s. Abb. 4.2.6) als Ausgangspunkt gewählt. Hier werden die beiden Vorschläge (s. Abb. 4.2.5) gemacht, die sich lediglich durch die PLZ unterscheiden. Der Benutzer kann jetzt seine Angaben genauer spezifizieren indem er eines der Einträge wählt und die Auswahl bestätigt. Sobald der Benutzer dieses getan hat, werden die Angaben in der Form für die Umgebungssuche vervollständigt (s. Abb. 4.2.6).

Abb. 4.2.6 Vervollständigtes Formular der mobilen Anwendung – Angaben für die Umgebungssuche

Der Benutzer entschied sich für den ersten Eintrag der Adressliste (s. Abb. 4.2.5). Durch mehrmaliges Zurückklicken gelangt man wieder zu den Angaben, die nach der Adressauswahl ergänzt wurden. Im Vergleich zur Abb. 4.2.3 wurde in der Abb. 4.2.6 die PLZ eingetragen.
Sobald die beiden Formulare abgeschickt wurden und die Adressangaben korrekt sind bzw. korrigiert wurden, kann sich der Benutzer die Umgebungskarte anzeigen lassen oder direkt zur Bootsliste klicken. Das entsprechende Auswahlmenü zeigt die Abbildung 4.2.7.

B. Anwendungsfall – Boot suchen
Dieser Anwendungsfall spiegelt den zentralen Bereich des Systems, nämlich der Suche nach Booten wieder, die für den Benutzer von Interesse sein könnten.

Szenario: Boot suchen

- Benutzer startet die Applikation,
- füllt ein Formular mit entsprechenden Parametern aus und schickt es ab.

- Dann kann er sich für eine Anzeige der Umgebungskarte entscheiden (dieses Szenario wird weiter unten beschrieben), oder er wählt die Bootsuche, die durch diesen Anwendungsfall beschrieben wird.
- Nach der Entscheidung für die Bootsuche zeigt das System dem Benutzer eine Liste mit Ausleihangeboten zur Auswahl an. Die Einträge der Liste sollten nach dem Preis sortiert sein.
- Der Benutzer wählt einen Eintrag aus und bekommt erneut ein Auswahlmenü zu sehen, das die folgenden Auswahlmöglichkeiten bietet. Er kann sich die Bootsdetails anzeigen lassen oder eine Routenberechnung durchführen lassen.
- Nach der Entscheidung für die Detailanzeige, bekommt der Nutzer die Bootsdetails zu sehen. Das weitere Vorgehen im Fall der Entscheidung des Nutzers für die Routenberechnung wird im Anwendungsfall *Routenberechnung* weiter unten im Abschnitt E dieses Kapitels beschrieben.
- Er hat hier die Möglichkeit das Boot zu reservieren oder zur letzten Auswahlliste zurück zu gelangen.

Im Folgenden wird der Anwendungsfall bildlich dargestellt.

Abb. 4.2.7 Das erste Auswahlmenü der mobilen Anwendung

Wenn die Angaben aus Abb. 4.2.2 und 4.2.3 verwendet werden und der Benutzer im obigen Auswahlmenü (s. Abb. 4.2.7) die Bootsliste auswählt, werden Boote (s. Abb. 4.2.8) aufgelistet, die den Benutzerangaben genügen.

4.2.8 Die Bootsliste

Die Boote sind in der Liste mit ihrem Namen und dem Preis pro Ausleihtag vertreten. Der Benutzer kann sich hier für eines der Boote entscheiden und sich weitere Details ansehen, die Route berechnen lassen oder letztendlich das Boot reservieren (siehe dazu Abb. 4.2.9). Die Entscheidung an dieser Stelle ist deshalb noch nicht bindend. Der Benutzer kann jeder Zeit wieder zu dieser Bootsliste zurückkehren und sich für ein anderes Boot entscheiden.

Abb. 4.2.9 Das Bootmenü

Die Bootdetails sind organisatorisch bei diesem Anwendungsfall als Ergebnis der Bootsuche untergebracht, weil es sich dabei um eine einfache Ausgabe handelt. Das nächste Bild (Abb. 4.2.10) zeigt die Details der beiden Boote aus der Bootsliste (s. Abb. 4.2.8).

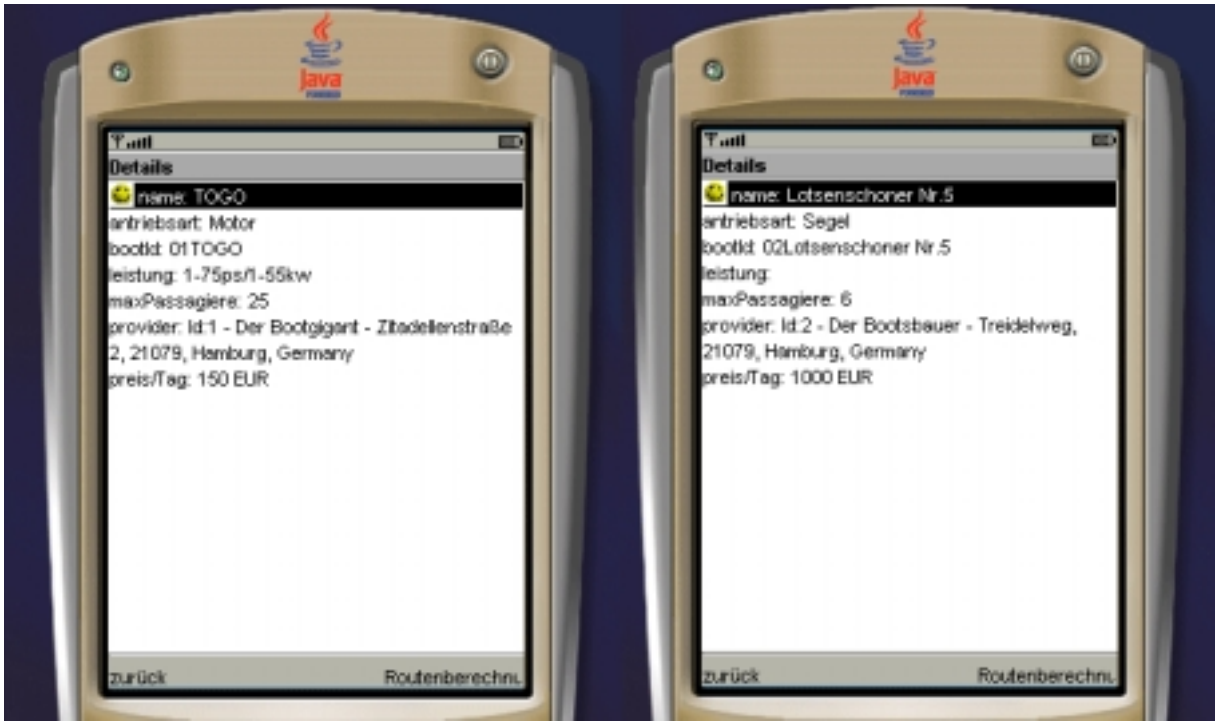

Abb. 4.2.10 Die Bootdetails der gefundenen Boote

Die Bootdetails beschreiben die Eigenschaften eines Bootes, das ausgeliehen werden kann.

Hier taucht als erstes der *Name* des jeweiligen Bootes auf.

Die *Antriebsart* kann die Ausprägung *Motor, Segel* oder *Motor* und *Segel* annehmen je nachdem, wie das Boot angetrieben wird.

Die nächste Ausgabe ist eine *BootId*. Einen solchen Bootidentifikator muss ein Benutzer nicht unbedingt sehen. Es sei denn es handelt sich um einen Systementwickler, der für die Weiterentwicklung zuständig ist. Weitere Einzelheiten zur Identifikation von Objekten, werden im Designkapitel beschrieben.

Die *Leistung* gibt die Leistungsklasse an, der das jeweilige Boot zugeordnet wurde.

maxPassagiere ist die maximale Kapazität eines Bootes bezüglich der Personenanzahl.

Der *Provider* eines Bootes ist bei der Ausgabe mit seiner Id, seinem Namen und seiner Adresse vertreten. Mit diesen Informationen kann der Benutzer, unter Benutzung der „zurück-Tasten“, wieder zur Umgebungskarte schalten. Dann kann er sich anschauen, welcher Provider das gewünschte Boot anbietet, und wo sich dieser befindet.

Preis/Tag gibt den Preis pro Ausleihtag an.

C. Anwendungsfall – Boot reservieren

Vorbedingung:
Die Funktionalität der folgenden Szenarien kann in der Anwendung erst dann aufgerufen werden, wenn ein bestimmtes Boot ausgewählt wurde. Der Aufruf soll bereits nach der

Auswahl eines Angebotes und in der Maske mit Bootdetails ermöglicht werden. Siehe dazu vorherigen Anwendungsfall – Boot suchen.

Szenario: Boot reservieren

- Der Benutzer befindet sich im Kontext eines ausgewählten Bootes und wählt *reservieren*.
- Der Dienst des Bootsproviders prüft ob das Boot bereits reserviert wurde. Diese Prüfung wurde bereits bei der Suchanfrage durchgeführt, jedoch könnte in der Zeit zw. der Suchanfrage und dem jetzigen Zeitpunkt eine Reservierung durch einen anderen Nutzer vorgenommen worden sein.
- Das Boot wird beim Bootsprovider reserviert.
- Der Benutzer erhält daraufhin eine Bestätigungsnachricht über die erfolgreiche Reservierung.
- Die Details über die Reservierung werden angezeigt. Diese müssen für eine erneute Einsicht vorgemerkt werden. Diese Vormerkung ist für den Nutzer wichtig, falls er sich diese Informationen noch mal in Erinnerung rufen möchte. Es handelt sich um Details wie z.B. die Adresse der Ausleihstelle oder Zeitangaben für die Ausleihe und Abgabe.

Das nächste Szenario ist eine Ausnahmebehandlung, welche die Interaktionsfolge beim Versuch einer Reservierung beschreibt. Dabei wurde das gewünschte Boot inzwischen von einem anderen Nutzer reserviert.

Ausnahmeszenario: Boot ist bereits reserviert

- Der Benutzer befindet sich im Kontext eines ausgewählten Bootes und wählt *reservieren*.
- Der Dienst des Bootsproviders prüft ob das Boot bereits reserviert wurde.
- Das Boot ist bereits reserviert, kann somit nicht mehr verliehen werden. Der Benutzer erhält einen aussagekräftigen Hinweis.
- Daraufhin bekommt er entweder die zuletzt erstellte Bootsliste mit Ausnahme des Fehlerverursachenden Eintrags zu sehen oder das Funktionsmenü mit der Möglichkeit eine erneute Suche zu starten.

D. Anwendungsfall - Routenberechnung
Für diesen Anwendungsfall soll ein Hauptszenario realisiert werden. Dem Benutzer soll die Berechnung der Route zw. einer durch den Benutzer vorgegebenen Adresse und dem Verein (bzw. der Ausleihstelle) angeboten werden.

Vorbedingungen:
Der Benutzer muss die Startadresse angegeben haben.
Der Provider bzw. dessen Adresse muss der Clientsoftware bekannt sein.

Ein Provider ist dem Client erst dann bekannt, wenn mindestens ein Boot des jeweiligen Providers bekannt ist. Somit sagt die letzt genannte Vorbedingung aus, dass die Routenberechnung erst dann aufgerufen werden kann, sobald die Suchfunktion (s. Anwendungsfall *Boot Suchen*) mind. ein Ergebnis geliefert hat und ein bestimmtes Boot ausgewählt wurde. Es muss dafür gesorgt werden, dass der Benutzer erst dann den Aufruf tätigen kann. Der Menüeintrag des Aufrufs könnte beispielsweise erst nach der Auswahl eines Bootes erscheinen.

Szenario: Route zw. der Benutzervorgabe und dem Verein (bzw. der Ausleihstelle)

- Benutzer ruft die Routenberechnung auf
- kann anschließend zw. grafischer und textlicher Routenbeschreibung wählen

Wenn die Vorbedingungen erfüllt wurden, stünden die beiden Endpunkte, nämlich der Startpunkt und der Endpunkt der Route, fest. Das Ergebnis einer Routenberechnung zeigt die folgende Abbildung (Abb. 4.2.11).

Abb. 4.2.11 Ergebnis einer Routenberechnung

Als Startpunktadresse wurde die Seehafenstrasse in Hamburg eingestellt (s. Abb. 4.2.11 und Abb. 4.2.3). Den Endpunkt repräsentiert der Provider eines ausgewählten Bootes.

E. Anwendungsfall – Dienstleistung publizieren

Der Provider publiziert seine Dienstleistung. Publizieren bzw. Registrieren bedeutet in diesem Fall die Übergabe der eigenen Adressdaten und der Dienstbeschreibungsdaten an den Verzeichnisdienst.

Die Funktionalität dieses Anwendungsfalls wird der Prototyp enthalten. Auf eine grafische Benutzungsschnittstelle wird jedoch verzichtet. Unter anderem deshalb, weil dies in den Aufgabenbereich des Entwicklers der Providersoftware gehört. Trotzdem wird hier von einer GUI ausgegangen.

Szenario: Dienst publizieren

- Der Provider oder sein Systemadministrator ruft das Programm auf.

- Anschließend füllt er ein Formular mit Parametern und Eigenschaften des Dienstes aus und schickt dieses ab.
- Das System fragt nach ob weitere Dienste registriert werden sollen. Wenn das der Fall ist, wird erneut das Formular zum Ausfüllen angeboten. Dabei werden einige Daten wie z.B. die Adresse des Verleihs vom letzten Formular übernommen. Es ist zu beachten, dass bei mehreren Dienstleistungen auch der Provider selbst, mehrmals eingetragen wird.
- Die Dienste werden beim Verzeichnisdienst registriert.

F. Anwendungsfall – Umgebungskarte anzeigen
Dieser Anwendungsfall ist erst im Nachhinein, d.h. während einer erneuten Analyse der Anforderungen erkannt und hinzugefügt worden. Der Benutzer sollte eine Möglichkeit bekommen, durch die alle Provider in der für ihn relevanten Umgebung angezeigt werden. Dies kann er tun nachdem er seine Wünsche bez. der Ausleihe in den Formularen spezifiziert hat. Eine Kartendarstellung scheint für diesen Zweck die geeignete Darstellung zu sein. Der Benutzer bekommt dadurch einen guten Überblick der Umgebung.

Vorbedingungen:
Die Adresse für den Bezugspunkt muss bekannt sein.

Szenario: Umgebungskarte anzeigen

- Der Benutzer Startet das Programm auf dem mobilen Gerät,
- füllt ein Formular mit Angaben zur Ausleihe und bestätigt,
- als Nächstes macht er Angaben zur Umgebungssuche (d.h. Bezugspunkt und Suchradius), schickt die Angaben ab,
- dann kann er sich zwischen einer Bootssuche und der Umgebungsanzeige entscheiden,
- wählt die Umgebungsanzeige,
- und erhält eine Karte mit eingetragenen Positionen der Provider als Displayausgabe.

Eine mögliche Umgebungskarte kann aus der folgenden Abbildung (Abb. 4.2.12) entnommen werden.

Abb. 4.2.12 - Umgebungskarte auf mobilem Gerät

Den relevanten Stellen in der Karte, können grafische Symbole zugeordnet werden. Einen Provider erkennt man z.B. an einem Ankersymbol, den Ausgangspunkt der Umgebungssuche an einem Stecknadelsymbol.

Für den Fall, dass keine Provider in der Umgebung gefunden werden, bekommt der Benutzer eine entsprechende Meldung in der Karte eingeblendet. Einen solchen Fall zeigt die nächste Abbildung (Abb. 4.2.13).

Abb. 4.2.13 Ausnahmeergebnis einer Umgebungssuche der Komponente *Umgebungskarte*

In diesem Fall wurde der Suchradius auf 100m eingestellt. Die Suche liefert keine Provider aus der relativ kleinen Umgebung. Trotzdem soll die Karte angezeigt werden, damit sich der Benutzer trotz fehlender Provider in der Umgebung orientieren kann.

4.3 Analyse – fachliche Systemkomponenten

„Als Software-Komponenten werden ausführbare Software-Module mit wohl definierter Schnittstelle bezeichnet“ - Zitat aus dem Buch von Prof. Kahlbrandt. Diese Definition trifft eher auf technische Komponenten zu, wie sie im Designkapitel beschrieben werden. Den technischen Komponenten liegen jedoch fachliche Komponenten zugrunde. Die fachlichen Komponenten werden innerhalb dieses Kapitels beschrieben. Hiermit soll verdeutlicht werden, was mit dem Einsatz der fachlichen Komponenten bezweckt wird.

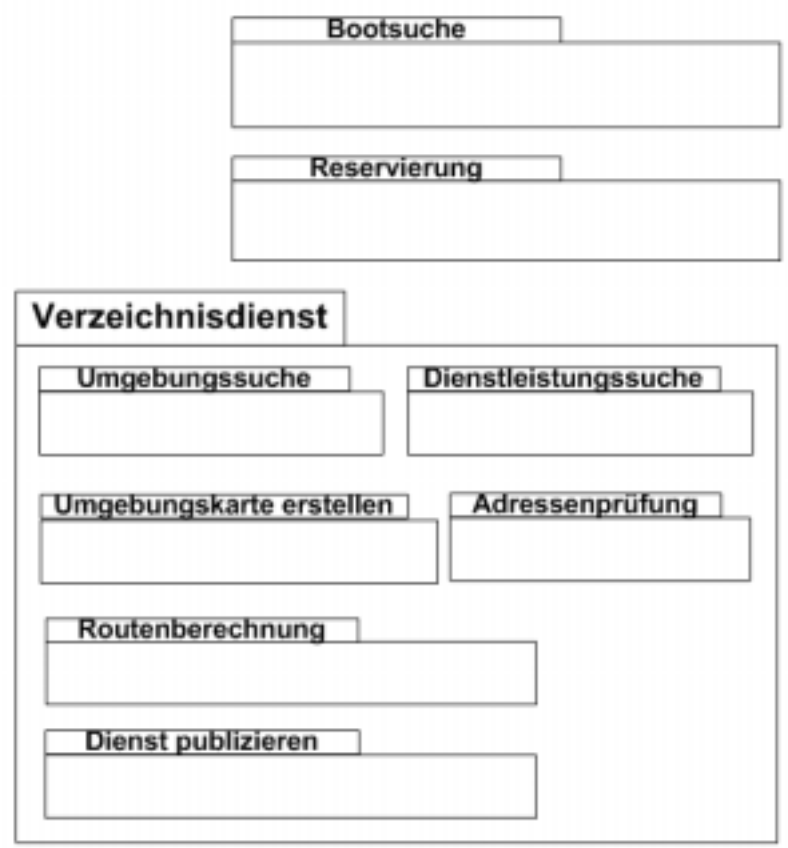

Abb. 4.3.1 Darstellung der Systemkomponenten

Die Abbildung 4.3.1 zeigt die funktionalen Komponenten des zu entwickelnden Systems. Die einzelnen Komponenten werden nun vorgestellt. Sie beschreiben die fachliche Funktionalität des Systems.

Bootsuche
Es soll nach Booten gesucht werden, die für den Benutzer der Anwendung in Frage kommen könnten. Die eigentliche Suche findet beim Bootsverleiher statt, da sich dort der Datenbestand befindet. Dabei werden beim Bootsprovider einige Prüfungen angestoßen, unter anderem die Prüfung nach Verfügbarkeit.

Reservierung
Der Benutzer soll die Möglichkeit erhalten, das gewünschte Boot zu reservieren. D.h. nach einer erfolgreichen Suche entscheidet sich dieser für ein bestimmtes Boot und äußert den Wunsch dieses zu mieten. Das Boot wird für einen bestimmten Zeitraum reserviert. Es muss dafür gesorgt werden, dass dieses in diesem Zeitraum nicht mehr von anderen Benutzern reserviert werden kann. Diese Prüfung musste bereits bei der Suche berücksichtigt werden, muss bei der Reservierung jedoch wegen der sich ständig ändernden Aktualität der Daten wiederholt werden. Bei der Entwicklung des Prototyps werden jedoch lediglich Schnittstellen zur Verfügung gestellt, und die tatsächliche Reservierung den Providern selbst überlassen. So fällt beispielsweise auch die Zeitraumprüfung in den Aufgabenbereich eines Providers.

Ortsbasierter Verzeichnisdienst
Bei einem Verzeichnisdienst spricht man von einem *Repository*, das Dienste verschiedener Anbieter verwaltet und die Suche nach diesen Diensten ermöglicht. Die Dienste müssen vorher beim Verzeichnisdienst registriert bzw. publiziert werden, deswegen handelt es sich dabei auch um eine so genannte *Registry*.

Unser Verzeichnisdienst verfügt zusätzlich über ortsbasierte Fähigkeiten. Diese Fähigkeiten finden sich in den Komponenten *Umgebungssuche*, *Umgebungskarte erstellen* und *Routenberechnung*.
Die ortsbasierten Fähigkeiten des Verzeichnisdienstes können nur dann genutzt werden, wenn die ortsbezogenen Angaben korrekt sind. Für die Prüfung der Adresse des Ausgangspunktes ist die Komponente *Adressenprüfung* verantwortlich. Neben dem Ausgangspunkt, der vom Benutzer angegeben wird, gehören auch die Dienstleistungen zu den ortsbezogenen Daten. Die Sicherstellung der korrekten Eingabe dieser Daten liegt im Verantwortungsbereich der Komponente *Dienst publizieren*.
Zu den Komponenten, die einen Verzeichnisdienst ausmachen, gehören die Komponenten „Dienstleistungssuche“ und „Dienst publizieren“.

Die geographische Grundlage des Verzeichnisdienstes soll ein aus Kapitel 2.1 resultierendes Geographisches Informationssystem bilden.
Die für das System relevanten Komponenten des Verzeichnisdienstes sind folgende:

Adressenprüfung
Diese Komponente wird zur Umsetzung des ersten Anwendungsfalls *Adressen prüfen* benötigt. Sie hat zur Aufgabe, die Benutzerangaben zum Ausgangspunkt zu überprüfen.
Es könnte nämlich vorkommen, dass eine Adressangabe nicht eindeutig ist. In diesem Fall würde die Komponente dem Benutzer alle potenziellen Adressen, die als mögliche Kandidaten in Frage kommen, zur Auswahl anbieten. Ein Beispiel dafür wurde bei der Beschreibung des Anwendungsfalls genannt. Außerdem könnte der Fall eintreten, wo gar keine Adresse gefunden werden kann, die den Angaben des Benutzers entsprechen würde. Auch diesen Fall muss die Komponente entsprechend behandeln können und dem Benutzer mit einer geeigneten Meldung über den Vorfall informieren.

Umgebungssuche
Vor der eigentlichen Suche nach Booten müssen die Bootsprovider bzw. deren Schnittstellen bekannt sein. So beinhaltet die Bootsuche eine *Suche nach Diensten* von Providern der Umgebung. Eine *Umgebungssuche* kann sich als sehr effizient erweisen besonders dann, wenn sich relativ viele Dienstanbieter (Bootsprovider) außerhalb dieser Umgebung befinden und von Vorne weg ausgeschlossen werden können. So lassen sich viele unnötige entfernte Aufrufe der Providerdienste verhindern.

Dienstleistungssuche
Einige dieser Angaben werden dafür verwendet die Anzahl anzufragender Provider zu reduzieren. Dazu soll die Art des Publizierens von Dienstleistungen genutzt werden. Dienstanbieter (in diesem Fall Bootsprovider) dürfen nämlich ihre Dienstleistungen über Attribute beschreiben, wenn sie diese beim Verzeichnisdienst registrieren (bzw. publizieren). Diesbezüglich wird der Verzeichnisdienst ein Attribut *Antriebsarten* vorsehen, welches die einzelnen Dienstleistungen hinsichtlich der Antriebsarten der angebotenen Boote beschreiben lässt. Nehmen wir mal an, dass der Benutzer nach Segelbooten sucht und dementsprechend auch als Antriebsart (s. obige Grafik) *Segel* gewählt hat. So würden Dienste, die beispielsweise nur Boote mit einem Motorantrieb anbieten, bereits auf der Ebene der Dienstleistungssuche aussortiert werden. Diese Attributeinschränkung bei der Dienstleistungssuche kann bzw. wird sich in den meisten Fällen positiv auf die Performanz des Systems auswirken. Ähnlich wie die Umgebungseinschränkung, die in einem Formular spezifiziert wird.

Umgebungskarte erstellen
Das Ziel ist es, dem Benutzer einen Überblick darüber zu verschaffen, welche und wie viele Provider sich in der gewünschten Umgebung befinden und somit bei einer eventuellen Bootsuche mit einbezogen werden. Solch eine Funktionalität kann eine Komponente bieten, die alle Provider kennt und ortsbezogene Informationen verwaltet. Es wird somit ein ortsbasierter Verzeichnisdienst bzw. ein spezielles geographisches Informationssystem sein, das hier zum Zuge kommt. Im besten Fall wird diese Komponente auch die Kartenerzeugung übernehmen. Für die Realisierung des Diplomarbeitsprojektes wird ein geographisches Informationssystem eines Fremdanbieters eingesetzt. Derartige Systeme wurden im Kap. 2.1 bereits ausführlich behandelt. Die Bootsprovider sollen von einem GIS in einer Benutzerdefinierten Datenbank verwaltet werden. Im Umfeld der GI-Systeme würde man die dort gespeicherten Providerinstanzen als so genannte POIs (s.Glossar) bezeichnen. Für die grafische Darstellung einer Umgebungskarte wird auf die ortsbezogen gespeicherten (POIs) zurückgegriffen (s. Abb. 4.2.12).

Routenberechnung
Die Routenberechnung schafft einen fließenden Übergang vom virtuellen d.h. softwareseitigen Dienstangebot zur tatsächlichen Nutzung des Produktes in der realen Welt. Der Nutzer der Software bekommt nämlich eine Beschreibung, wie er die Ausleihstelle mit einem Auto erreichen kann. Im Vorfeld einer Routenberechnung müssen einige Arbeiten durchgeführt werden. Die eigentliche Berechnung benötigt z.B. keine Adressangaben sondern Angaben, die für eine Berechnung verwendet werden können. Und zwar sind das die Weltkoordinaten, die eine bestimmte Stelle der Erde relativ genau beschreiben. Die Adressangaben müssen deshalb in Weltkoordinaten umgewandelt d.h. geokodiert werden. Diese sehr spezielle Funktionalität gehört unter anderen zum Spezialgebiet der bereits vorgestellten geografischen Informationssysteme. Außer der Geokodierung werden noch andere Dienstleistungen des Fremdanbieters genutzt. Im Bezug auf die Routenberechnung gehört die Erzeugung der Routenkarte zu den ausgelagerten Funktionalitäten.

Dienst publizieren
Das Publizieren muss vom Dienstanbieter initiiert werden und entweder direkt oder indirekt beim Verzeichnisdienst beantragt werden. Falls alle notwendigen Angaben korrekt sind und vom Verzeichnisdienst akzeptiert werden, wird der Dienst mit seiner Beschreibung dem Datenbestand des Repository hinzugefügt. Im Falle eines ortsbezogenen Verzeichnisdienstes werden die Adressangaben zusätzlich noch geokodiert (s.Glossar).

4.4 Architektur

In diesem Kapitel wird der Überblick über die gesamte Architektur des angestrebten Systems skizziert. Der Sinn dieses Kapitels ist die Erläuterung der logischen und physischen Struktur. Um solche Strukturen aufzubauen, werden im Laufe der Analyse und des Entwurfes Überlegungen bezüglich der Funktionalität, der Wiederverwendbarkeit, der Performance, der Nutzbarkeit, Robustheit und Sicherheit des hier zugrunde liegenden Systems angestellt und die geeigneten Entscheidungen getroffen. Sowohl wirtschaftliche wie auch die technisch machbaren Randbedingungen müssen berücksichtigt werden.

Architektur – Vorschlag einer Systemarchitektur
In diesem Abschnitt wird eine Architektur vorgeschlagen, welche die im vorherigen Kapitel (Kap. 4.3) vorgestellten Systemkomponenten berücksichtigt.
Wie bereits bei der Betrachtung der Systemkomponenten erwähnt wurde, werden auf den Clients lediglich Webserviceaufrufe umgesetzt. Die eigentliche Anwendungslogik wird über

mehrere Server verteilt. Damit sind ein systemeigener Server und mehrere externe Server gemeint.
Die erste Überlegung war soviel an Funktionsumfang wie möglich von den Clients auf einen Proxy zu verlegen, da die Clients bekanntlich über relativ wenig Speicherplatz verfügen. Ein Proxy wird als vermittelnde Einheit zwischen die Clients und die eigentlichen Zielserver geschaltet. In diesem Fall gibt es mehrere Zielserver. Es sind die Server der Bootsprovider und der MS-Mappointserver. Sobald eine Clientanfrage kommt verarbeitet der Proxyserver diese und schickt die Antwort an den Client zurück.

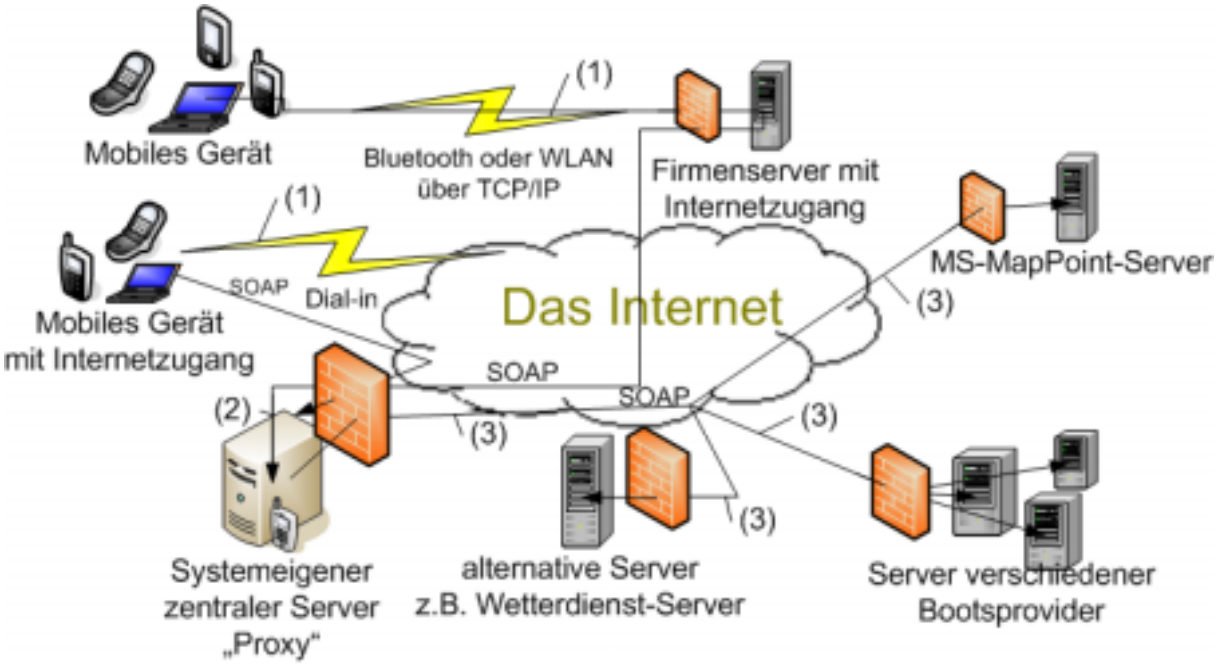

Abb. 4.4.1 Die physikalische Systemarchitektur

Damit die Clients über den Firmenserver ins Internet gelangen, müssen sie diesen als Standardgateway in ihrer Netzwerkeinstellung eingetragen haben.
Es spielt keine Rolle ob die Clients direkt oder indirekt ins Internet gelangen. Die Anwendung wird in beiden Fällen funktionieren. Voraussetzung ist jedoch, die Unterstützung von TCP/IP.

Architektur – Alternative Architekturen für die Realisierung
Bei der Realisierung unseres Projektes wird eine vereinfachte Architektur verwendet.
Das soll bedeuten, dass einige Systemeinheiten bzw. deren Funktionalität, wie das mobile Gerät, der *Proxy*, der Internetserver und die Server der Bootsverleiher, die unter reellen Bedingungen auf mehreren Servern verteilt sein würden, sich nun auf einer einzigen gemeinsamen Hardware befinden. Diese Maßnahme wird den Aufbau der Testumgebung vereinfachen. Die Kommunikation bleibt die gleiche wie in der für die reale Umgebung vorgeschlagenen Architektur. Die Vereinfachung wird durch die folgende Abbildung (Abb. 4.4.2) verdeutlicht.

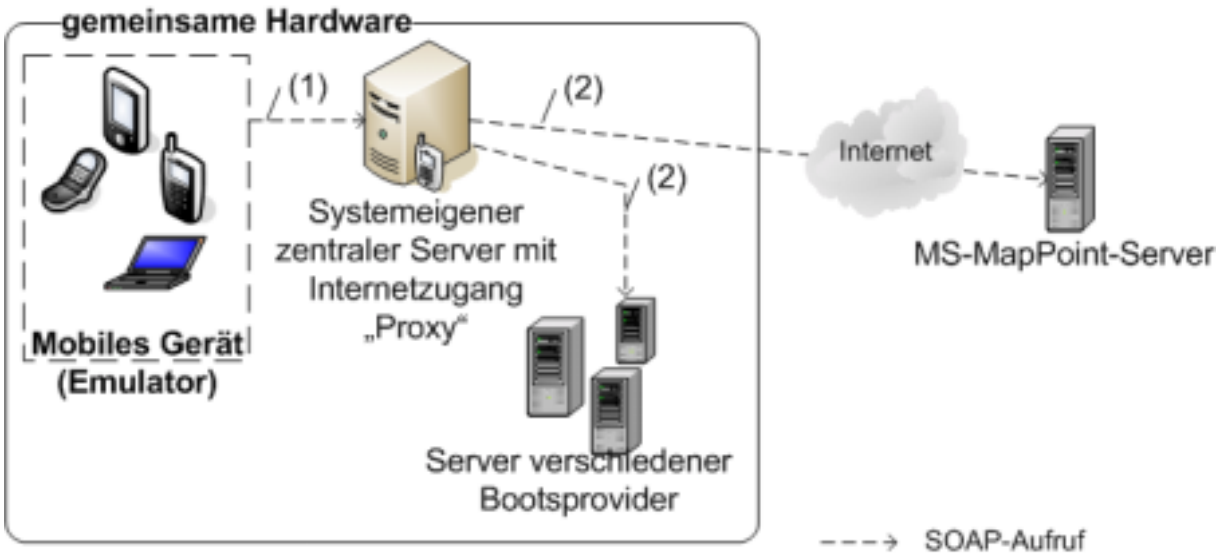

Abb. 4.4.2 Vereinfachte Systemarchitektur

Für die Vereinfachung der Architektur durch eine gemeinsame Hardware, kommen selbstverständlich nur die Systemeinheiten in Frage, die im Rahmen der Arbeit entstanden sind. Bei der gemeinsamen Hardware handelt es sich um einen Testrechner, der die gesamte systeminterne (in der Abb. 4.4.2 als gemeinsame Hardware zusammengefasst) Architektur simulieren wird. Übrige, externe Systemeinheiten, wie der MS-MapPoint-Server, existieren bereits auf Servern anderer Organisationen und sind nur über definierte Schnittstellen erreichbar. Die Server der Bootsprovider (Bootsverleiher) sind externe Server, ähnlich wie der MapPoint-Server. Dennoch wird auch deren Funktionalität zu Test- und Demonstrationszwecken auf der gemeinsamen Hardware simuliert.

4.5 Fazit

In diesem Kapitel „Analyse“ wurden Anforderungen an das System spezifiziert. Aufgrund von diesen Anforderungen sind Anwendungsfälle entstanden, die einen Einblick in die Interaktion zwischen dem System und dem Benutzer des Systems erlauben. Die für die Umsetzung der Anwendungsfälle benötigte Funktionalität wurde funktionalen Komponenten zugeordnet und beschrieben. Als letztes wurden Systemarchitekturen vorgestellt, die für die Realisierung des Systems in Frage kommen. Dabei wurde eine Architektur besprochen, die unter realen Bedingungen aufgebaut werden könnte und eine, die für den Prototypen verwendet wird.
Als nächstes sollte der Leser erfahren, wie die einzelnen funktionalen Komponenten aus 4.3. auf diese Architektur aus 4.4. verteilt werden und wie sie technisch realisiert werden können. Das alles erfährt der Leser im nächsten Kapitel Design.

Kapitel 5. Design

Das Analysemodell des Analysekapitels, welches das System aus der funktionalen Benutzersicht darstellt, wurde, aus der Sicht eines Systementwicklers, auf einer relativ hohen Abstraktionsebene beschrieben. Die funktionalen Komponenten müssen anschließend implementiert werden. Die Implementierung ist dagegen sehr technisch und enthält viele Details, die das Verständnis der Funktionsweise des Systems erschweren würden. Das Designkapitel führt deshalb eine Abstraktionsebene, nämlich die technischen Komponenten dazwischen.
Kap. 5.1 liefert Informationen über allgemeine Überlegungen und Designentscheidungen, die beim Entwurf des Systems entstanden sind. Dazu gehören Überlegungen zu den clientseitigen Aufrufmodellen und den zu transportierenden Objekten. Die fachlichen Komponenten des Analysemodells werden im Kap. 5.2 auf die Systemarchitektur verteilt. Im Kap. 5.3 werden die bereits erwähnten technischen Komponenten eingeführt. Dieses Unterkapitel ist das umfangreichste und zugleich das relevanteste Kapitel des Designkapitels, denn dort werden die wichtigsten technischen Funktionalitäten des Systems erläutert. Im Kap. 5.4 werden nicht-funktionale Anforderungen, wie die Sicherheit und Robustheit des Systems, angesprochen. Die Sequenzdiagramme im Kap. 5.5 verstärken das Verständnis des dynamischen Verhaltens des Systems. Im Kap. 5.6 wird ein Resümee des Designkapitels gemacht.

5.1 Designmuster/ Designentscheidungen

Das Fassade-Muster

Durch den blackboxartigen Aufbau des Systems wird das Informationhiding unterstützt. Dies ist eine der Forderungen an eine SOA. In einer SOA sollen die Services technisch neutral bleiben was heißen soll, dass an der Schnittstelle keine Interna des implementierenden Systems gezeigt werden. Ein derartiger Aufbau entspricht dem Fassade-Muster.

Designentscheidungen für den mobilen Client

Beim clientseitigen Design musste eine Entscheidung getroffen werden, ob eine dauerhafte Speicherung des Systemzustands notwendig ist.
Clientseitig kann eine Reservierung z.B., erst nach der gezielten Auswahl eines Bootes durch den Benutzer vorgenommen werden. Es liegt nahe, nach einer Auswahl, den Client zu veranlassen, sich das Boot zu merken, um eventuell zum späteren Zeitpunkt (z.B. nach einem erneuten Start der Applikation) eine Reservierung vorzunehmen. Wenn man aber bedenkt, dass sich die Angebote ständig ändern, so wird es sinnvoller eine erneute Suche anzustoßen.
Man könnte aber auch direkt vor einer Reservierung eine Verfügbarkeitsprüfung durchführen lassen. Doch würden dem Nutzer dadurch eventuell neue, bessere Angebote vorenthalten bleiben, die nur bei einer „frischen Bootssuche“ zum Vorschein kommen würden.
Falls sich in der Praxis erhebliche Performanceprobleme ergeben sollten, könnte in diesem Sinne ein Kompromiss gefunden werden.
Die Detailangaben zum ausgesuchten Boot müssen jedoch zumindest für eine anschließende Routenberechnung erhalten bleiben. D.h. sie müssen clientseitig gespeichert werden. Bei einer serverseitigen Speicherung müsste der entsprechende Webservice eine zusätzliche Schnittstelle offen legen. Der Overhead an Kommunikation wäre in diesem Fall größer als der clientseitige Speicherplatzverbrauch für die wenigen Detailinformationen eines Bootes. Die Details lassen sich aber auch temporär im Speicher halten, so, dass eine persistente Speicherung nicht nötig ist. Die Verantwortung dafür liegt bei der *BootMidlet*-Komponente. Beim letzteren Vorgehen, welches letztendlich für die Umsetzung verwendet wurde, muss man mit einem Datenverlust rechnen, der jedoch in diesem Fall vertretbar ist. Bei einem

eventuellen Stromausfall oder Systemabsturz würde der gesamte Systemzustand verloren gehen und damit auch die Information, welches Boot ausgewählt wurde.

Clientseitige Aufrufmodelle

Im Analysekapitel wurde über Schnittstellenharmonisierung bzw. Schnittstellenintegration gesprochen. Das Ergebnis dieser Überlegungen war, dass sich die Bootsverleiher an bestimmte Konventionen halten müssen. Sie müssen z.B. ihre Methoden an bestimmte Signaturen anpassen, vorgegebene Parametertypen verwenden und einiges mehr. Diese Maßnahmen mussten vorgenommen werden, damit auf dem zentralen Server alle Bootsverleiher bzw. deren Webservices und Objekte gleich behandelt werden können und dadurch die Komplexität des zentralen Servers minimiert wird.
Nachdem Überlegungen hinsichtlich des Schnittstellenaufbaus gemacht wurden und durch Konventionen festgehalten werden, sollte darüber nachgedacht werden wie ein Webservice über ein solches Interface aufgerufen werden kann. Es wird dabei zwischen dem statischen Aufrufmodell (static binding) und dem dynamischen Aufrufmodell (dynamic binding) unterschieden.
Bei statischen Modellen wird das WSDL-Dokument aus dem serverseitigen Programmcode durch einen Generator erzeugt. Der Clientseitige Code wird durch einen WSDL-Compiler aus dem WSDL-Dokument generiert. Der Cliententwickler kann anschließend den generierten Programmcode, den so genannten Stub, in die Clientanwendung integrieren.
Im dynamischen Modell gibt es Programmierbibliotheken, durch die ein WSDL-Dokument inspiziert und ein Aufruf des Webservice zur Laufzeit konstruiert werden kann. Eine bekannte Lösung, die rein dynamisch ist, stellt das WSIF (Webservice Inspection Framework) dar. Aber auch das JAX-RPC (Java API for XML RPC) lässt sich dynamisch verwenden.

Es gibt drei Alternativen im Bezug auf entfernte Aufrufe im Bereich von Webservices. Eine davon ist statischer Natur (stubbasierter Aufruf). Eine andere Möglichkeit bilden so genannte Dynamische Proxies. Der Einsatz von DII (Dynamic Invocation Interfaces) ist die dritte Alternative für einen entfernten Aufruf. Bei allen drei Methoden muss das WSDL-Dokument zur Entwicklungszeit bekannt sein. Damit ist nicht nur die URL, sondern auch der Inhalt des Dokuments gemeint.

Beim *stubbasierten Aufruf* werden zur Entwicklungszeit aus der WSDL Javaklassen generiert – so genannte Stubs, wobei der verantwortliche Compiler explizit parametrisiert und aufgerufen werden muss. Das kann als Nachteil im Bezug auf die Bequemlichkeit gewertet werden. Dafür kann der Entwickler die Objekte und Methoden sehen und bedienen, was wiederum eine Bedienungserleichterung ist. Als ein weiterer Vorteil, kann die automatische Erzeugung von Typklassen aus den WSDL-Metadaten gewertet werden. Noch ein Vorteil von Stubs ist der, dass der zur Entwicklungszeit generierte Code keine Zeit mehr zur Ausführungszeit für die Compilierung benötigt.

Die Laufzeitumgebung der Dynamic Proxies übersetzt die WSDL erst zur Laufzeit in Javacode. Das Ergebnis sind so genannte Dynamic Proxies. Der Vorteil gegenüber der ersten Lösung ist, dass keine Stubs zur Entwicklungszeit generiert werden müssen, was eine vereinfachte Bedienung bedeutet. Die Methoden der Proxies werden bereits im Clientcode verwendet, was die Bedienung z.B. gegenüber der DII erheblich vereinfacht. Dafür müssen das Interface und die benutzerdefinierten Typklassen bereits zur Entwicklungszeit auf dem Client vorliegen.

Das *DII* ermöglicht das dynamische Konfigurieren eines Aufrufs und letztendlich den Aufruf des Webserviceendpoints. Konfigurieren bedeutet hier ein manuelles Typemapping. Dazu

werden außer der Standarddatentypen auch benutzerdefinierte Datentypen, die in der WSDL als XML Schema Typen beschrieben werden, abgebildet. Das Konstruieren des Aufrufs geschieht erst zur Laufzeit. Die Typklassen werden nicht automatisch erzeugt, wie es bei der Stub-Lösung der Fall ist, und müssen deshalb bereits zur Entwicklungszeit auf dem Client vorliegen. Eine entsprechende Laufzeitumgebung muss das DII implementieren.

Das Axis Framework ist eine Servlet basierte JAX RPC Runtime und unterstützt laut Spezifikation alle drei Methoden, insbesondere die Stub-Lösung, die sich für den Einsatz auf dem Proxy am besten eignet. Axis wurde in dieser Diplomarbeit serverseitig verwendet.

Auf den mobilen Clients wird kSOAP verwendet, wo nur DII möglich ist. Auch in kSOAP muss der Inhalt des WSDL-Dokuments der Clientsoftware bekannt sein, damit das Type-Mapping durchgeführt werden kann.

Transportierte Objekte
Mit transportierten Objekten sind die Objekte gemeint, die zwischen Client und Server über das Netz ausgetauscht werden. Dazu gehören die Boot-Objekte, Adressen-Objekte und die AusleihSpezifikation-Objekte. Diese Objekte müssen, sobald sie auf dem Client ankommen, von den Marshallern aus dem XML-Strom ausgepackt und in eine neue Objekthülle verpackt werden. Der Anwender von kSOAP muss die entsprechenden Marshaller selber bauen, weil das Marshalling benutzerdefinierter Objekte kein Bestandteil des kSOAP-Frameworks ist. Objekte werden by-value und nicht by-reference übertragen, daher muss deren Identifikation zwingend aufgrund von fachlichen Schlüsseln erfolgen.

Herstellung der Eindeutigkeit von Objekten
Sowohl die Bootobjekte als auch die Providerobjekte entstehen bei den Providern (Bootsverleihern) selbst. Die Vergabe der Objekt-Ids (Objektidentifikatoren) ist im Falle der Providerobjekte für den Bootsverleiher optional. Die *Providerobjekte* werden auf jeden Fall mit einer neuen Id ausgestattet, sobald sie beim Verzeichnisdienst registriert werden. Die registrierten Provider müssen bereits in der Datenbank des Verzeichnisdienstes eindeutig identifizierbar sein. Sie bekommen deshalb eine fortlaufende Nummer, die auch nach dem Herausholen der Objekte aus der Datenbank, beibehalten wird. Die Software des Bootsproviders bekommt die neue Id selbstverständlich mit, und kann diese im eigenen Datenbestand aktualisieren. Die Vergabe neuer Ids ereignet sich auf dem zentralen Rechner und zwar sofort nach dem Erhalt der Providerobjekte vom jeweiligen Provider. Dadurch ist eine saubere Schnittstelle für die Erzeugung der Ids gegeben, wodurch vor allem gefährliche Inkonsistenzen bei der Id Vergabe verhindert werden.

Die Vergabe neuer Schlüssel für Bootobjekte verfügt nicht über eine saubere Schnittstelle, das hat jedoch einen triftigen Grund. Der Grund für den Kompromiss ist, wie so oft in der Softwareentwicklung, ein erheblicher Gewinn an Performanz.
Der Kompromiss besteht darin, dass die Boote nach dem sie vom Provider verschickt wurden, erst auf dem mobilen Gerät über einen fachlichen Schlüssel verfügen werden. Dort müssen die Bootobjekte wegen der Deserialisierung auf jeden Fall angefasst werden. Deshalb wird der fachliche Schlüssel erst hier vergeben und nicht auf dem zentralen Server. Bei n Booten spart man sich damit den n-maligen Zugriff auf die Objekte auf dem Proxy. Beim Aufbauen des *Bootobjektes* durch den entsprechenden Marshaller bekommt das Objekt den neuen Schlüsselwert. Damit ist die fachliche Eindeutigkeit erst innerhalb der Clientsoftware gegeben. Das ist kein Problem, solange der Proxy lediglich die Vermittlerrolle übernimmt. Dieser verarbeitet die Objekte nicht und braucht deshalb auch kein Boot eindeutig zu identifizieren. Im Falle der Übermittlung von Bootobjekten trifft genau das zu. Durch den

Erhalt der BootId, die vom Provider vergeben wurde bleibt auch die serverseitige Identität erhalten. Die Zusammensetzung und der Gültigkeitsbereich des fachlichen Schlüssels kann der Abbildung Abb.5.1.1 entnommen werden.

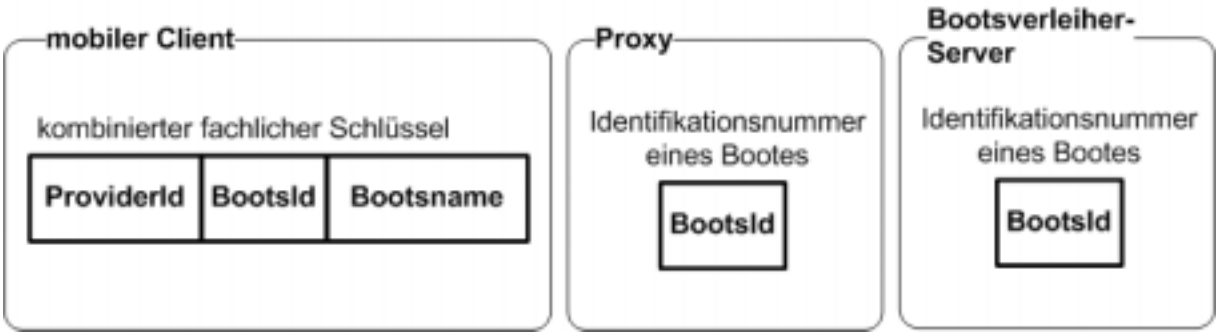

Abb. 5.1.1 Fachliche Identität eines Bootobjektes im System

Der kombinierte fachliche Schlüssel setzt sich aus der Identifikationsnummer des Providers (ProviderId), der BootId und dem Bootsnamen zusammen. Für die Herstellung der Identität würde die Kombination aus der ProviderId und dem Bootsnamen ausreichen. Die Bootid wird jedoch auch mitgespeichert, weil sie für die Bootreservierung beim Provider benötigt wird. Der Proxy wird im Falle der Reservierung die Umwandlung des fachlichen Schlüssels in die, für den Bootsverleiher verständliche, BootsId übernehmen. Hierdurch bekommt der Bootsverleiher nichts von der Umwandlung mit und hat somit eine relativ saubere Schnittstelle vor sich.

5.2 Verteilung der funktionalen Komponenten auf die Systemarchitektur

Die funktionalen Komponenten des Bootsverleihsystems wurden im Analysekapitel eingehend behandelt. Diese sollen nun auf die Systemarchitektur verteilt werden. Auch die Systemarchitektur wurde im Analysekapitel vorgestellt.

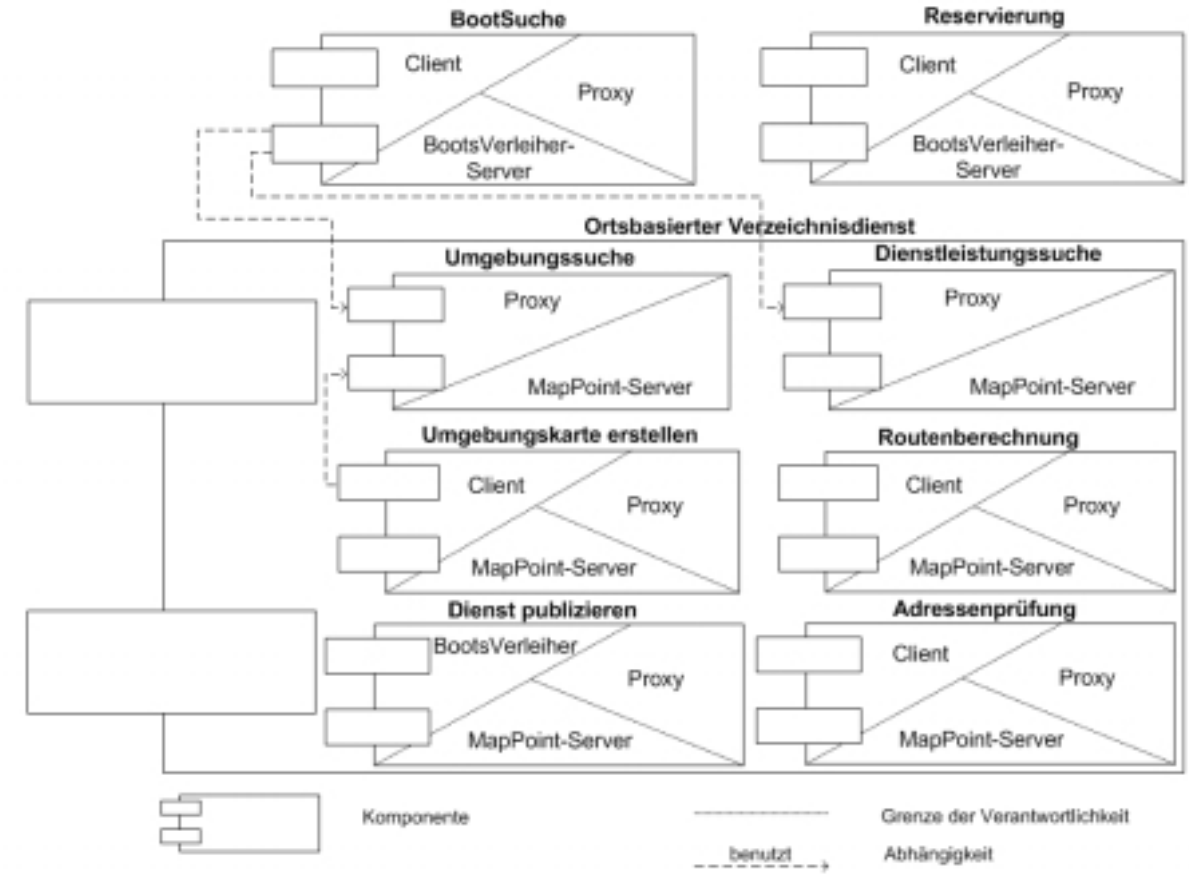

Abb. 5.2.1 Abbildung der funktionalen Komponenten auf die Hardwarekomponenten der Systemarchitektur

Der mobile Client ist als *Thin Client* konzipiert, auf Grund dessen die eigentliche Anwendungslogik über einige Server verteilt sein wird. Dabei handelt es sich um den systemeigenen Server *Proxy* und mehrere externe Server, wie den *MapPoint-Server* und die *Server der Bootsverleiher* (s.Kap.4.4-Architektur).
In der Abbildung 5.2.1 erkennt man die acht funktionalen Komponenten aus dem Analysekapitel wieder, die in mehrere Teile aufgeteilt wurden. Die Aufteilung der Komponenten bedeutet eine Aufteilung der Verantwortung für die Komponententeile. Die Verantwortung wird auf die Systemeinheiten der Systemarchitektur aus Kap. 4.4 verteilt. Die Systemeinheiten sind der mobile Client, der Proxy, der MapPoint-Server und die Server der Bootsverleiher.
Der ortsbasierte Verzeichnisdienst umfasst all die Komponenten, die etwas mit Daten zu tun haben, die direkt an einen Ort gebunden sind. So ist bei der Komponente der Dienstleistungssuche und bei der Komponente, die für das Publizieren des Dienstes zuständig ist, die Dienstleistung an einen Ort gebunden. Bei der Umgebungssuchkomponente und der Komponente, die für die Erstellung der Umgebungskarte zuständig ist, ist es der vom Benutzer angegebene Ausgangspunkt welcher ortsgebunden ist. Die Komponente der Routenberechnung hat mit zwei ortsbezogenen Datensätzen zu tun, nämlich dem Start- und dem Zielpunkt der Route. Diese Ortsbezogenheit ist allen Komponenten der Verzeichnisdienstkomponente gemeinsam, genauso wie die Verbindung zum MapPoint-Server, der die ortsbezogenen Daten verwaltet.
Auf den Clients werden lediglich Webservices des Proxy aufgerufen. Der Proxy bearbeitet die Anfragen indem er die meisten an externe Server weiter leitet. Der zentrale Server wurde *Proxy* genannt, weil er die Funktion eines Proxy-Servers übernimmt d.h. die Weiterleitung der Clientanfragen. Der Proxy unseres Systems schlüpft dafür in zwei Rollen. Einerseits ist er der

Server, der die Clientanfragen entgegennimmt und bearbeitet. Andererseits muss er selber Clientanfragen bei externen Servern stellen, um die Anfragen, die er als Server angenommen hatte, bearbeiten zu können. Diese Doppelrolle wird nach der Einführung einer niedrigeren Abstraktionsebene, nämlich der technischen Komponenten besser verständlich. Deshalb verweise ich an dieser Stelle auf die Beschreibung des *MietServices* als technische Komponente des Proxys im Abschnitt 5.3 Design - Technische Komponenten.
Dafür kann der MietService beim MapPoint-Server ortsbezogene Informationen abfragen, wie es bei drei Komponenten des Verzeichnisdienstes der Fall ist. Gemeint sind die Komponenten *Routenberechnung*, die *Umgebungskarte erstellen* und *Adressen prüfen*. An allen drei Komponenten sind der mobile Client, der Proxy und der MapPoint-Server beteiligt.
Die Doppelrolle des Proxy wird auch bei den Komponenten *BootSuche*, *Reservierung* und *Dienst publizieren* sichtbar. Bei den zwei erstgenannten werden die Anfragen des mobilen Clients vom Proxy an die Server der Bootsverleiher weitergeleitet. Die dritt genannte Komponente wird nicht wie gehabt vom Client sondern vom Bootsverleiher initiiert und an den MapPoint-Server weitergeleitet.
Die Komponenten *BootSuche* und *Reservierung* sind nicht Bestandteil des Verzeichnisdienstes. Sowohl die Suche nach Booten, als auch die Reservierung finden auf den Servern der Bootsverleiher statt, wobei auch hier der zentrale Server als Proxy benutzt wird. Die Komponente *BootSuche* verwendet zwei Komponenten des Verzeichnisdienstes – die Dienstleistungssuche und die Umgebungssuche, um die Bootsuche entsprechend einschränken zu können. Diese zwei Komponenten werden somit nur indirekt vom mobilen Client verwendet, weshalb in der Komponentengrafik (Abb. 5.2.1) der Client innerhalb der Komponenten nicht aufgeführt wurde. Die Komponente *Umgebungskarte erstellen* verwendet die Komponente *Umgebungssuche*, um die Provider der Umgebung in die Karte eintragen zu können.

Gründe für die Einführung des Proxys
Die Systemarchitektur sieht einen zentralen Server als Proxy zwischen den mobilen Clients und den externen Servern vor. Da stellt sich die Frage, weshalb der Client die externen Server nicht direkt anspricht.
Es gibt mehrere Gründe für den Einsatz des Proxy.
Der Proxy wurde in erster Linie aus dem Grund der Lastverteilung eingeführt. Die mobilen Clients weisen nur geringe Speicherressourcen und eine begrenzte Prozessorleistung auf, wären deshalb mit der gesamten Clientsoftware überlastet. Ohne den Einsatz des Proxy müsste das mobile Gerät die Schnittstellen für die Zugriffe auf die Bootsverleihserver und auch die für den Zugriff auf den MapPoint-Server tragen. Außerdem müsste es die gesamte Geschäftslogik enthalten. Der Proxy übernimmt die Weiterleitung der Serviceanfragen und enthält zusätzlich den Großteil der Geschäftslogik. Dadurch wird der mobile Client entlastet. Durch diese Entlastung des mobilen Gerätes erhöht sich die Spannweite der Geräteklassen, die für den Einsatz der Software verwendet werden können. Somit wird der an das System gestellten Anforderung, möglichst hoch portabel zu sein, genüge getan.
Der nächste Grund, der für den Proxy spricht ist, dass eine zentrale Servereinheit benötigt wird. Sie wird vor allem aus Sicherheitsgründen gebraucht. Im Bezug auf die Sicherheit soll die Registrierung der Bootsverleiher und ihrer Dienste über eine zentrale Stelle erfolgen. Ein direkter Zugriff der Bootsverleiher auf die Datenbank des MapPoint Servers würde, aufgrund der Art des Zugriffs, eine große Sicherheitslücke ergeben. Für den Zugriff muss nämlich die gesamte Datenbanktabelle auf den lokalen Rechner herunter geladen werden und kann erst dann manipuliert werden.

5.3 Design - Technische Komponenten

Bisher war von funktionalen Komponenten die Rede, nun werden die technischen Komponenten besprochen, die eine Verbindung zwischen dem abstrakten Analysemodell und der konkreten Implementierung herstellen.
Der Komponentendefinition (aus Kap. 4 Analyse-Komponenten) nach, können die Services der Provider als auch der zentrale Webservice *MietService* und die MIDlets der mobilen Clients als Komponenten betrachtet werden. Diese Komponenten sollen in diesem Kapitel aus technischer Sicht erläutert werden. Die nächste Abbildung veranschaulicht, wie die Verteilung der technischen Komponenten auf die Systemarchitektur vorgenommen wurde. Außerdem wurden Abhängigkeiten zwischen den Komponenten eingezeichnet, an denen der Leser erkennen kann, wie die Kommunikation zwischen den Komponenten funktioniert. Dabei wurde lediglich die Richtung der Anfragen berücksichtigt, jedoch nicht die der Antworten.

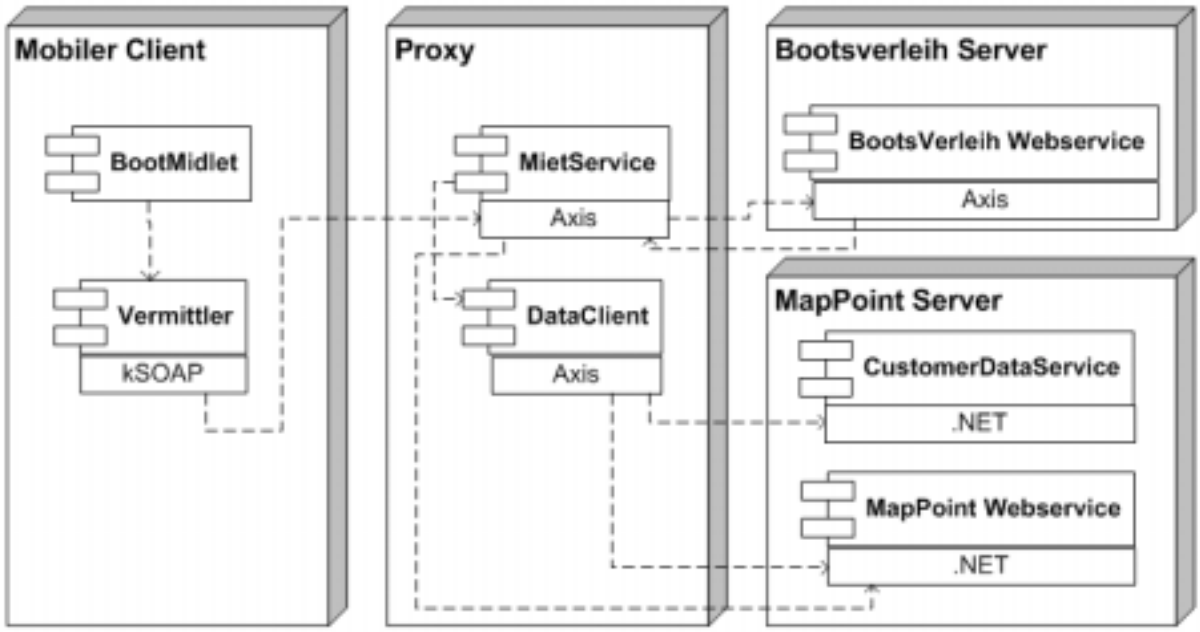

Abb. 5.3.1 Verteilung technischer Komponenten auf die Systemarchitektur

Die Komponenten, *Vermittler*, *MietService*, *DataClient*, *BootsVerleih-Webservice*, *CustomerDataService* und *MapPoint-Webservice* sind für die Webservicezugriffe zuständig. Für die Übermittlung der Anfragen und Antworten verwenden diese das Internet oder lokale Netzwerke als Medium. Die zu übermittelnden Daten müssen entsprechend des Webservice-Standards serialisiert werden, damit diese auf der anderen Seite richtig interpretiert werden können. Diese Aufgaben übernehmen spezielle Webservice Frameworks. Für den mobilen Client wurde das Framework kSOAP vorgesehen, für den Proxy und den zu implementierenden Bootsverleihserver das AXIS-Framework. Der MapPoint-Server verwendet das .NET-Framework. Die nächste Grafik (Abb.5.3.2) verdeutlicht noch einmal die Kommunikation unter Verwendung der Frameworks.

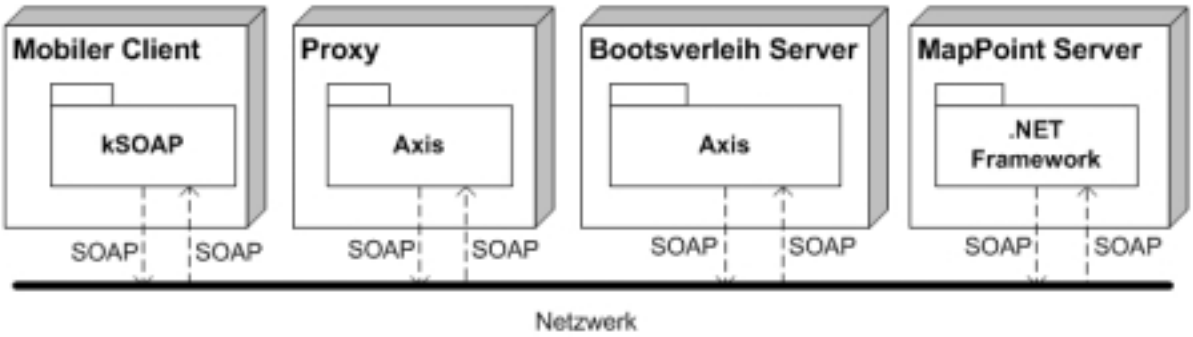

Abb. 5.3.2 Einsatz von Webservice-Frameworks

In der obigen Abbildung (Abb. 5.3.2) sieht man, dass sich die einzelnen Systemeinheiten, mit Unterstützung der Frameworks, über ein gemeinsames Medium (Internet oder lokale Netze) unterhalten können. Aufgrund der Verwendung des Standardprotokolls (SOAP) können sich diese auch untereinander verstehen.

Technische Komponenten des mobilen Clients

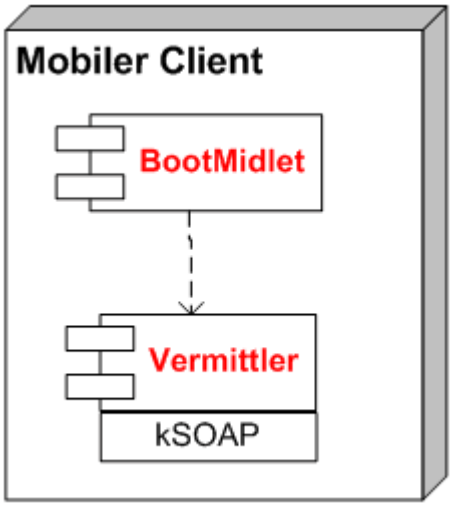

Abb. 5.3.3 Technische Komponenten des mobilen Clients

Das ***BootMIDlet*** ist die zentrale Komponente der Clientsoftware.
Mithilfe dieser Komponente ist der Benutzer in der Lage, direkt mit dem System zu interagieren. Sie baut eine grafische Benutzungsoberfläche auf dem mobilen Client auf.

Zum Erstellen von GUIs (graphical user interface) unter J2ME (s.Kap 3.2.B – Java auf mobilenGeräten) gibt es in der MIDP-API das lcdui-Package (javax.microedition.lcdui).
Eine wichtige Rolle für unsere Anwendung spielt die Klasse *List* aus dem oben genannten lcdui-Package. Objekte dieser Klasse sollen die Rolle eines Menüs übernehmen und Methodenaufrufe als Einträge beinhalten. Beim Auswählen eines dieser Einträge wird ein bestimmter Anwendungsfall (s.Kap 4.2) abgearbeitet. In folgender Abbildung werden die Klassen des lcdui-Packages aufgeführt.

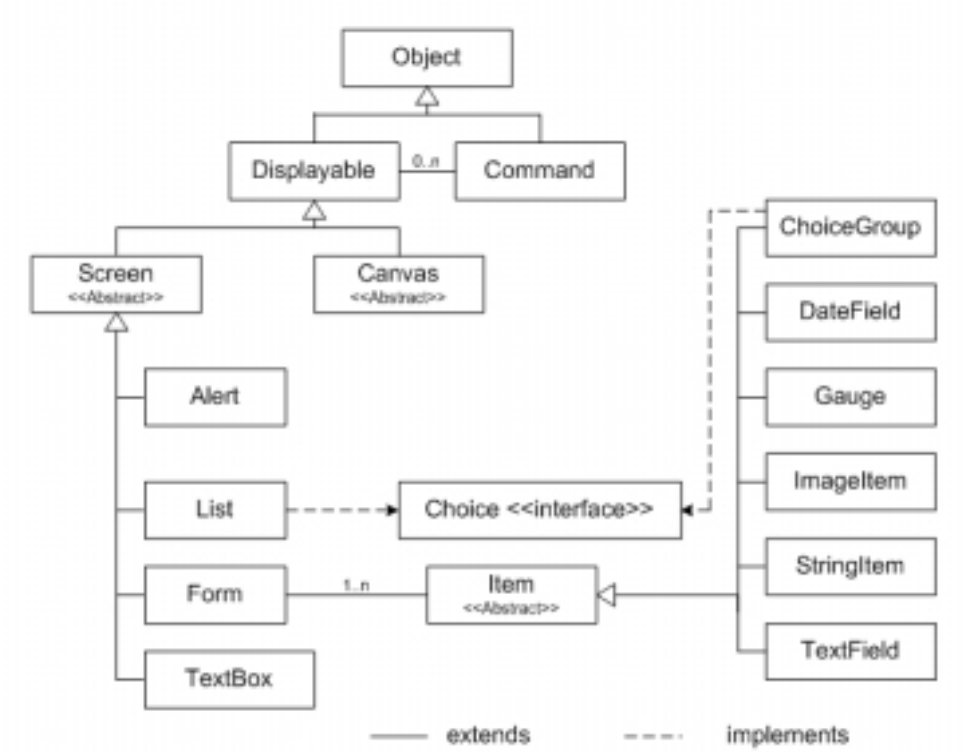

Abb. 5.3.4 Klassen des lcdui-Packages

Bei allen Klassen des obigen Diagramms, außer der Canvas-Klasse, handelt es sich um Highlevel-Klassen. Die Canvas erlaubt es, eine Graphik auf einer niedrigen Ebene (Lowlevel) zu manipulieren. Sie wird überwiegend für die Entwicklung von Handyspielen verwendet. Die Erläuterung der einzelnen Klassen des lcdui-Packages findet der Leser im Anhang dieser Arbeit.

Die zweite technische Komponente - ***Vermittler*** des mobilen Clients, realisiert all die fachlichen Komponenten des Analysemodells, an deren Ausführung der mobile Client beteiligt ist (s. Abb.5.3.5): Die *BootSuche*, *Reservierung*, *Umgebungskarte erstellen*, *Routenberechnung* und *Adressenprüfung*.

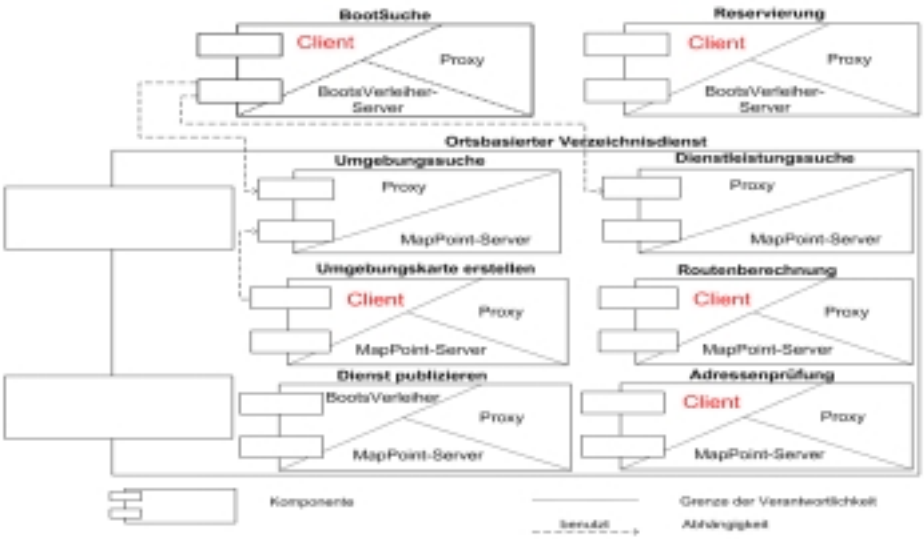

Abb. 5.3.5 Funktionale Komponenten, an denen der mobile Client beteiligt ist

Die Vermittler-Komponente übernimmt all die Aufgaben des mobilen Clients, die mit der Webservicekommunikation zu tun haben. Die technischen Details der Kommunikation sind dadurch gekapselt und für andere Komponenten nicht sichtbar. Die Komponente *BootMidlet* äußert lediglich den Wunsch eines bestimmten Methodenaufrufs. Der Vermittler übernimmt dann die Kontrolle darüber. Die Vermittlerkomponente sorgt für das Verschicken und Empfangen von SOAP-Nachrichten, Typemapping und De/Serialisierung der Nachrichten. Eine genauere Beschreibung dieser Abläufe geben die Sequenzdiagramme im Abschnitt 5.5 des Designkapitels.

Technische Komponenten des Proxy

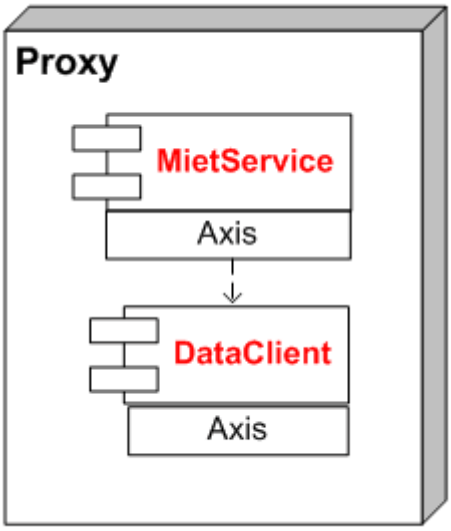

Abb. 5.3.6 Technische Komponenten des Proxy

Der Mietservice

Gründe für den Einsatz eines Proxy-Servers wurden bereits zu Beginn des Designkapitels genannt (Kap. 5.2 im Abschnitt „Gründe für die Einführung des Proxy“).

Die zentrale Komponente des Proxy ist der ***MietService***, der als Webservice realisiert wurde. Diese Komponente ist auch die zentrale Komponente für das gesamte System, denn sie ist an der Realisierung aller fachlichen Komponenten beteiligt. Die Grafik 5.2.1 zeigt, dass der Proxy für jede fachliche Komponente einen Teil der Verantwortung trägt. In all diesen Fällen steckt der *MietService* als technische Realisierung dahinter.

Die technische Komponente *MietService* agiert in einer Doppelrolle, indem sie einerseits Anfragen mobiler Clients verarbeitet und andererseits selber Anfragen an externe Webservices stellt. Die Rollen des *MietServices* werden durch die folgenden zwei Grafiken (Abb.5.3.7 und Abb.5.3.8) visualisiert.

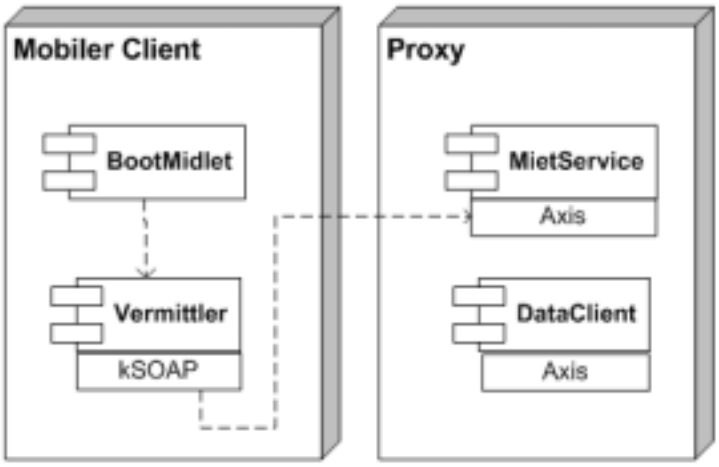

Abb. 5.3.7 Der MietService des Proxy in der Server- bzw. Serviceprovider-Rolle

Die Abbildung Abb.5.3.7 verdeutlicht das Verhalten des Proxy bzw. dessen MietServices in der Rolle des Serviceproviders. Die Komponente erhält eine SOAP-Anfrage und ist für deren Bearbeitung und das Verschicken einer entsprechenden Antwort an den Client zuständig.

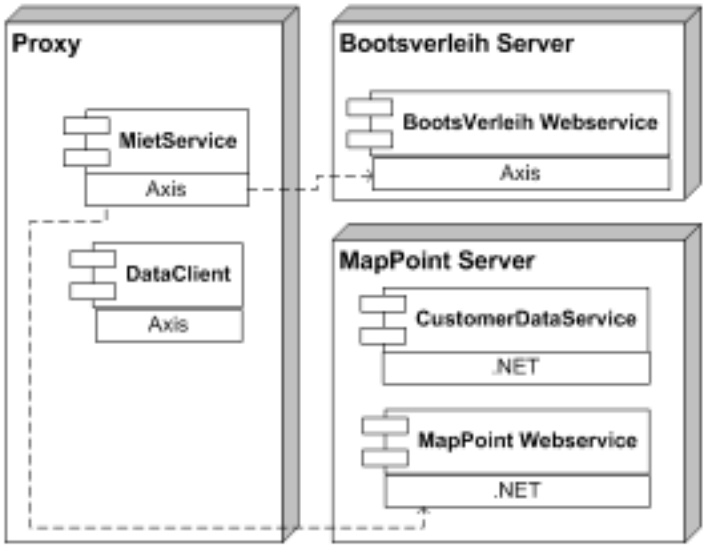

Abb. 5.3.8 Der MietService des Proxy in der Client- bzw. Servicerequester-Rolle

Die Abbildung Abb.5.3.8 verdeutlicht wiederum das Verhalten des Proxy bzw. dessen MietServices in der Rolle des Servicerequesters. Die Komponente in dieser Rolle muss selbst eine Clientanfrage bei den externen Webservices stellen, um Informationen für den Aufbau einer Antwort auf eine vorher erhaltene Anfrage eines mobilen Clients zu sammeln.

Der Dataclient

Eine weitere Komponente, die sich auf dem zentralen Server befindet ist für die Verwaltung der Daten (POIs) der benutzerdefinierten DB auf dem MapPoint-Server verantwortlich. Es ist der **DataClient**, der seine Aufgaben unter Verwendung des Customer Data Services (wurde weiter unten bei den Komponenten des MapPoint Servers untergebracht) erfüllt. Zu den wichtigsten Aufgaben der Komponente gehört es zu erfahren, welche Bootsverleiher sich bereits beim Mietsystem registriert haben. Dazu enthält die Klasse, die diese Komponente repräsentiert unter anderem eine Methode, die alle vorhandenen Providerobjekte liefert, die für die Erzeugung der *ProviderStubs* benötigt werden. Zur Zeit müssen die Stubs von Hand

angelegt werden. Somit ist diese Methode als Orientierung für den Administrator des Systems gedacht.
Für die Umsetzung dieser Funktionalität gibt es mindestens drei alternative Vorgehensweisen.

Providerermittlung mithilfe mehrerer Umgebungssuchen
Die erste Art der Providerermittlung geschieht mithilfe der FindNearBy-Funktion (wird weiter unten bei den technischen Komponenten des MapPoint-Servers erläutert) und des maximalen Suchradius von 400km. Sie ist deshalb räumlich begrenzt, lässt sich jedoch durch Kombination mehrerer FindNearBy-Abfragen erweitern (letztlich auch für die ganze Welt). Diese Art der Providerermittlung ist somit eine durchaus verwendbare Alternative, jedoch eine, die weitere Überlegungen hinsichtlich der Weltweiten suche erfordert.

Providerermittlung mithilfe des DataClients
Als die zweite alternative Vorgehensweise für eine weltweite Ermittlung der Provider d.h. des gesamten DB Inhalts (Provider sind in der MapPoint-DB als POIs abgelegt), musste eine andere Methode verwendet werden. Dafür würde man die dataDownload()-Methode des DataClient verwenden, die eine flache Datei mit dem gesamten DB-Inhalt produziert. Mit einer entsprechenden Auswertung der Datei würde man dann alle existierenden Provider ermitteln. Bei dieser Vorgehensweise würde man somit nach dem Download auf einer Kopie der Datenbanktabelle arbeiten, die anschließend wieder auf den Server hochgeladen werden müsste. Der größte Nachteil dieser Vorgehensweise ist jedoch der, dass sie nicht beliebig oft angewandt werden kann, da die Download- wie auch die Upload-Vorgänge des MapPoint-Servers als so genannte Jobs gewertet werden und in ihrer Anzahl pro Tag auf 50 begrenzt sind. Ein weiterer Nachteil ist der hohe Zeitverbrauch, der aufgrund mehrmaliger Speicherung der Datei auftritt.

Diese so genannten Jobs werden vor allem zum Registrieren bzw. Entfernen der Provider benötigt, weil die APIs des MapPoint-Servers keine geeigneten Schnittstellen dafür bieten. Bei diesen Vorgängen wird sowohl ein Download als auch ein Upload durchgeführt. So sind maximal 25 solcher Vorgänge bzw. Jobs am Tag möglich.

Die zweite Alternative erfasst in ihrer einfachen Fassung bereits die gesamte Welt, jedoch konkurriert sie mit den essentiellen Funktionalitäten des Systems, nämlich den zum Registrieren und Entfernen von Providern. Aus diesem Grund ist die erst genannte Vorgehensweise die bessere Alternative. Doch gibt es noch eine andere Möglichkeit der Providerermittlung.

Providerermittlung mithilfe einer Propertysuche
Die API des MapPoint-Webservices, der weiter unten bei den technischen Komponenten des MapPoint-Servers erläutert wird, bietet eine Methode an, die es ermöglicht nach Attributen der benutzerdefinierten POI-Datenbank zu suchen. Da jedoch nicht nach bestimmten Attributen gesucht werden soll, sondern alle Providerdatensätze (POIs) als Ergebnis erwartet werden, muss als Suchattribut ein Attribut angegeben werden, welches ohnehin bei jedem Providerdatensatz vorhanden ist. Ein solches Attribut ist für jede Datenbank des MapPoint-Servers vorgeschrieben und muss den Namen *ServiceCategory* tragen. Der Wert dieses Attributes ist bei jedem Datensatz der gleiche. Im Falle der Datenbank mit den Providerentities ist der Wert *Boote* eingetragen.
Die dritte Alternative wurde für die Realisierung gewählt, weil sie den Zweck vollständig erfüllt und keine weiteren Maßnahmen erfordert.

Vermeidung von Datenverlusten beim Registrieren
Aufgrund fehlender APIs seitens des MapPoint-Servers, geschieht sowohl das Registrieren als auch das Entfernen von Providern durch Manipulieren der MapPoint-Datenbank unter Verwendung der Methoden zum Downloaden und Uploaden. Im Mehrbenutzerbetrieb (mehrere Bootsverleiher) könnte es zu Datenverlusten kommen. Es müsste hier unbedingt ein Transaktionskonzept entwickelt werden. Zur Zeit wird das Problem umgangen, indem der *DataClient* zentral vom Administrator bedient wird.

Technische Komponenten des MapPoint Servers
Als Ergebnis des zweiten Kapitels hatte sich das MapPoint von Microsoft als geeignetes geographisches Informationssystem (GIS) herauskristallisiert.

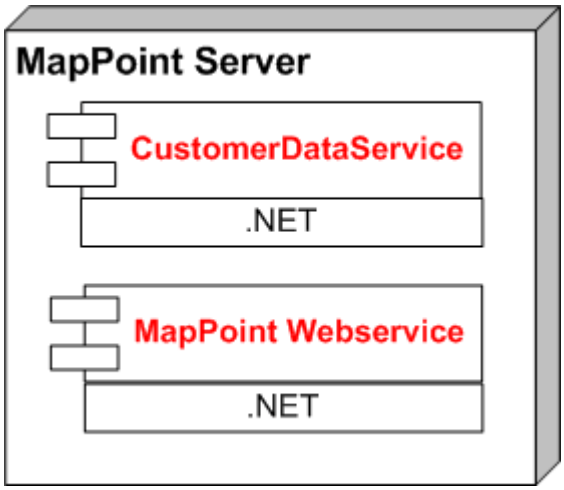

Abb. 5.3.9 Technische Komponenten des MapPoint-Servers

Der MapPoint Webservice ist eine der Komponenten des MapPoint-Servers. Sie stellt dem Programmierer vier Dienste zur Verfügung. Es sind dies der Common-Service, der Find-Service, der Render-Service und der Route-Service. Die Namensgebung verrät, dass jeder dieser Services einen anderen Anwendungsbereich des GIS abdeckt. Alle vier werden hier kurz vorgestellt.

Der Common-Service
Dieser Service der MapPoint-Webservice-API liefert allgemeine Informationen über die Datenbanken des MapPoint-Servers. Es lassen sich Informationen über die Fähigkeiten der einzelnen Datenbanken gewinnen. Die möglichen Fähigkeiten befinden sich in einer Enumeration, die den Namen *DataSourceCapability* trägt.

Zu den möglichen Capabilities gehört die Capability *CanDrawMaps*. Mit dieser Fähigkeit werden Datenbanken gekennzeichnet, die die meisten Methoden des Render-Service unterstützen.
CanFindPlaces erlaubt das Anwenden der Methode find() des Find-Services, die eine Suche nach Ortschaften erlaubt.
CanFindNearby unterstützt die anderen relevanten Methoden des Find-Services, die das Arbeiten mit bzw. das Suchen nach eigenen POIs ermöglichen.
CanRoute zeichnet die Datenbanken aus, die eine Routenkalkulation erlauben. Dabei sind die meisten Methoden des Route-Services gefragt.

CanFindAddress erlaubt das Anwenden der FindAdress-Methode des Find-Services auf eine Datenbank. Damit ist die Möglichkeit des Suchens nach bestimmten Adressen gegeben.
HasIcons verrät ob eine Datenbank Icons enthält. Icons sind kleine Bilder, die bestimmte Stellen bei einer Kartenausgabe kennzeichnen. Beispielsweise wird eine bestimmte Art von POIs mit Icons gekennzeichnet. Auch der Start und Endpunkt einer Route werden auf einer Karte mit Icons gekennzeichnet, die in diesem Fall auch als Pushpins bezeichnet werden.

Der Find-Service
Für die Realisierung der Providerermittlung des DataClients wurden bereits weiter oben mehrere alternative Vorgehensweisen vorgestellt. Beim ersten Blick in die MapPoint-Webservice-API schien die *find()*-Methode des Find-Service die richtige zu sein. Der Name der Methode klingt sehr allgemein und das war genau das wonach gesucht wurde, denn es sollten Provider ohne gefiltert zu werden aus der Datenbank herausgeholt werden. Die *find()*-Methode sucht jedoch nach bestimmten Plätzen und Ortschaften in den bestehenden Datenbanken des MapPoint-Servers, wie z.B. in der *MapPoint.EU* für Europa oder der *MapPoint.NA* für Nordamerika und nicht in den benutzerdefinierten privaten Datenbanken. Damit die *find*-Methode auf eine Datenbank angewendet werden kann, muss die Datenbank die Capability *CanFindPlaces* aufweisen. Dies trifft nur für die MapPoint spezifischen Datenbanken zu.
Die folgenden drei Methoden können nur auf den privaten benutzerdefinierten und POIs enthaltenden Datenbanken angewendet werden.
Unter Verwendung der *FindByID-Methode* lassen sich die Provider nach ihren identifizierenden Schlüsseln suchen.
Die Methode FindByProperty *ermöglicht die Suche nach Attributen der Entities. Diese Suchart hatte sich letztendlich für die Realisierung der Providerermittlung der DataClient-Komponente des Proxy durchgesetzt.*
FindNearRoute *findet die, einer Route nahe gelegenen, POIs.*

Das mächtigste Werkzeug des Find-Service und überhaupt der gesamten MapPoint-Webservice-Komponente ist die *FindNearby-Methode*. Mithilfe dieses Werkzeugs kann die funktionale Komponente *Umgebungssuche*, und damit eine der wichtigsten Eigenschaften einer ortsbasierten *Registry*, implementiert werden. Mit dem Einsatz eines Filters können ungewünschte Entities bereits auf dem MapPoint-Server aussortiert werden. Das vermindert die Netzbelastung und womöglich auch die anfallenden Verbindungskosten. Der Filter kann Entities unter anderem nach benutzerdefinierten Eigenschaften bzw. Attributen filtern. Filterausdrücke, so genannte *FilterExpressions*, erlauben das Angeben komplexer SQL ähnlicher Abfragen, die eine feinere Filterung verursachen. Mit diesen Fähigkeiten kann die fachliche Komponente *Dienstleistungssuche* sehr bequem und ebenfalls effizient (hinsichtlich der Netzkosten) implementiert werden.

Die folgende Grafik Abb.5.3.10 zeigt einen Codeausschnitt der FindNearBy-Methode, die die Dienstleistungssuche realisiert. Es wird nach Providern (Bootsverleihern) gesucht, die Boote mit einer bestimmten Antriebsart anbieten.

```
ArrayOfAnyType arrayOfAnyType = new ArrayOfAnyType();
Object[] myParameters = new Object[2];
String myFirstParameter = "Boote";
String mySecondParameter = spez.getAntriebsart();
myParameters[0] = myFirstParameter;
myParameters[1] = mySecondParameter;
arrayOfAnyType.setAnyType(myParameters);

String myText;
FilterExpression myFilterExpression = new FilterExpression();
//Wenn als Antriebsart "alle" vom Benutzer ausgewählt wurde - wird das
//Attribut nicht berücksichtigt und alle Provider werden durchsucht.
if(spez.getAntriebsart().equalsIgnoreCase("alle")){
    myText = "ServiceCategory=(0)";
//In allen anderen Fällen nur die Provider durchsuchen, deren Attributvalues mit den
//beutzerspezifizierten(durch "spez") übereinstimmen.
}else{
    myText = "SupportedModeOfDrive=(1)";
}

myFilterExpression.setText(myText);
myFilterExpression.setParameters(arrayOfAnyType);

FindNearbySpecification findSpec=new FindNearbySpecification();
findSpec.setDataSourceName("MagdalenaP113355.113355.Bootsprovider");
findSpec.setFilter(new FindFilter());
findSpec.getFilter().setEntityTypeName("Boote");
findSpec.getFilter().setExpression(myFilterExpression);
```

Abb. 5.3.10 Codeausschnitt aus route.MapBuilder.FindNearBy() – FilterExpression auf Antriebsart

Der Render-Service
Die Methoden wie *GetBestMapView*, *GetMap*, *GetLineDriveMap* lassen geographische Kartendarstellungen anfertigen, in die gewünschte POIs und Routen eingezeichnet werden können. Dieser Service wird zur Realisierung der funktionalen Komponente *Umgebungskarte anzeigen* und der ebenfalls funktionalen Komponente *Routenberechnung* beitragen.

Der Route-Service
Mit den Methoden *CalculateRoute* und *CalculateSimpleRoute* können Routen zwischen POIs und anderen geographischen Positionen, so genannten Locations, ermittelt werden. Auch dieser Service, ähnlich wie der *Render-Service*, trägt zur praktischen Umsetzung der fachlichen Komponente *Routenberechnung* bei.

Der Customer Data Service stellt eine weitere Komponente des MapPoint-Servers dar. Diese Komponente wird zum Verwalten der benutzerdefinierten Datenbank verwendet.
Im Folgenden wird der Ablauf der Speicherung eigener Daten mithilfe der Customer Data Service API beschrieben.
Das Uploaden der eigenen Datenbestände auf den MapPoint Server geschieht in drei Schritten.

Als erstes wird ein Upload-Job kreiert. Dies geschieht durch die Verwendung der Methode CustomerDataService.StartUpload.
Nach einem erfolgreichen Zugriff wird eine Job-ID zurückgegeben, die im weiteren Verlauf immer wieder gebraucht wird.
Eine *UploadSpecification* muss der ersten Methode übergeben werden. Diese enthält Parameter, die etwas darüber aussagen ob in der Produktions(production)- oder

Testumgebung(staging) gearbeitet wird. Außerdem wird die gewünschte Genauigkeit der Geokodierung (das *Geocoding-Match-Level* wie Straße, Stadt, Postleitzahl, Gebiet oder Region) in der Spezifikation angegeben, und ob nach dem Geokodieren entstandene mehrdeutige Felder ignoriert werden sollen. Für die Zwecke dieser Diplomarbeit wird der *Geocoding-Match-Level* auf *Street* (Strasse) gestellt, denn die Ermittlung der Erdkoordinaten der Stadtmitte oder sogar des Zentrums eines Postleitzahlenbereichs wäre zu ungenau. Falls eine Strasse nicht eindeutig bestimmbar sein sollte, muss die technische Komponente *Customer Data Service* einen Fehler melden. Dieses Verhalten ist beabsichtigt und muss entsprechend behandelt werden. Diese Aufgabe übernimmt die fachliche bzw. funktionale Komponente *Adressen prüfen* des Analysemodells. Fachliche Komponenten wurden auch im Designkapitel in der Abb. 5.2.1 abgebildet.

Im zweiten Schritt des Uploadvorgangs werden die Daten hochgeladen. Es handelt sich dabei um eigene Points-of-Interest oder um Polygondaten, die mithilfe der Methode CustomerDataService.UploadData auf den MapPointserver transportiert werden. Dabei müssen die Daten in Chunks aufgeteilt werden, falls deren Menge 1MB überschreiten sollte. Die job-ID, die beim ersten Schritt erhalten wurde, wird der Methode als Parameter übergeben.

Erst im dritten Schritt, sobald die Methode CustomerDataService.FinishUpload aufgerufen wurde, werden die Daten in eine Datenbank des MapPoint-Servers geladen und committed.

Der CustomerDataService fügt weitere sechs Felder hinzu. Die vier Felder *MatchCode, MatchedMethod, MatchedAddress, InputModified* gehören dazu und beinhalten Informationen über die geokodierten Ergebnisse. *EditedLocationUTC, EditedPropertyUTC* sind weitere zwei Felder, in denen die Zeit und das Datum der Änderung von Ortsangaben und der Eigenschaften durch die Tools des CustomerDataServices festgehalten werden.

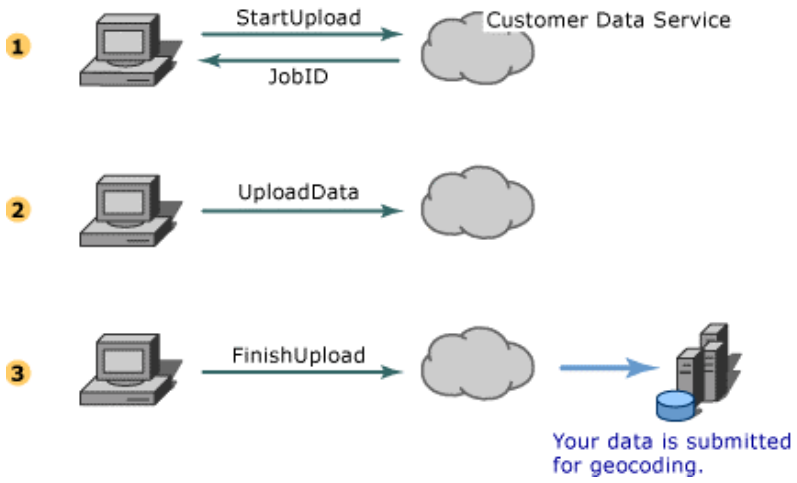

Abb. 5.3.11 Der Uploadvorgang - Übernommen von der Internetseite „msdn.microsoft.com“

Der Download eigener Dateien funktioniert etwas anders als ein gewöhnlicher Dateidownload im WWW. Der Customer Data Service lädt die Daten zuerst in eine Datei, die auf dem MapPoint-Server abgespeichert wird. Anschließend sendet er eine URL an den anfragenden Client zurück. Mit dieser URL, die zwei Wochen lang aktiv bleibt, kann auf die Datei zugegriffen werden. Der Zugriff auf die Datei erfolgt mittels eines sicheren (SSL/TLS) Kanals und nur unter Verwendung der Credentials. Mit Credentials sind hier die von Microsoft vergebenen Login-Daten gemeint. Das Downloaden geschieht ebenfalls in drei Schritten. Als erstes wird ein neuer Download-Job kreiert. Dafür wird die Methode

CustomerDataService.StartDownload verwendet. Auch hier erhält man eine JobID, die den Downloadjob identifiziert. Dann kann mit getDownloadFileURL(jobID) die Internetadresse geholt werden, mit deren Hilfe eine Datei auf dem lokalen Rechner erzeugt werden kann. Diese enthält dann die POIs aus der MapPoint Datenbank.

Serverkomponenten von Bootsverleihern

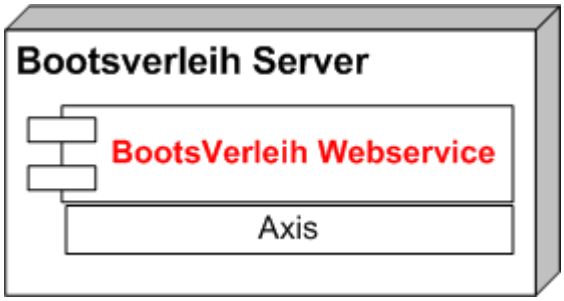

Abb. 5.3.12 Technische Komponenten des Bootsverleihers

Der Server eines Bootsverleihers enthält eine technische Komponente, den Bootsverleih Webservice. Als technische Komponente unterstützt der Webservice die Umsetzung der fachlichen Komponenten des Analysekapitels.
Die Komponente muss Schnittstellen für die Bootsuche und die Reservierung eines Bootes anbieten.
Die eigentliche Reservierung eines Bootes findet auf dem Server des Providers statt. Deshalb bleibt die Realisierung der Software diesem selbst überlassen. Ebenso muss der Provider für die Konsistenz seiner Daten sorgen. Wegen des Mehrbenutzerbetriebes sollten Vorgänge wie die Reservierung die ACID-Eigenschaften aufweisen. So kann der Datenbestand konsistent und zuverlässig gehalten werden.

5.4 Sicherstellen der Nicht-funktionalen Anforderungen

Definition [Kahlbrandt2001] „Durch nicht-funktionale Anforderungen werden Rahmenbedingungen beschrieben, unter denen die Funktionen erbracht werden müssen. Anforderungen die unabhängig von einem speziellen Anwendungsfall sind, oder die aus anderen Gründen nicht als Funktionen beschrieben werden können."

Sicherheit
Die Sicherheit der Daten auf dem MapPoint Server wird von Microsoft gewährleistet. Auch der Zugriff auf die dort gehosteten Daten über den Customer Data Service ist abgesichert. Dieser Service des MapPoint Servers führt kritische Vorgänge aus und kann deshalb nur über eine verschlüsselte Verbindung (https) erfolgen. Ein Mithören der Nachrichten sowie ein Eingriff in bestehende Verbindungen wird dadurch unterbunden.

Robustheit durch Sessionfreiheit
Ein Service sollte frei vom Session-Zustand sein, d.h. er darf kein Konzept einer User Session haben. Ein solcher Service kann idealer Weise sowohl von einem Dialog, wie von einer Batchanwendung gerufen werden.
Falls es gelingen sollte all diese Eigenschaften strikt umzusetzen, wird z.B. im Falle einer ungewollten Verbindungsunterbrechung, was bei Bluetooth- bzw. WLAN-Verbindungen

leicht passieren kann, lediglich eine neue Verbindung aufgebaut werden müssen. Ein Recovery des Systemzustands wäre dadurch überflüssig.

5.5 Sequenzdiagramme

Die Sequenzdiagramme sollen das dynamische Verhalten des Systems beschreiben und damit für ein besseres Verständnis der Anwendungsfälle des Analysekapitels sorgen. Die Anwendungsfälle wurden zu Gunsten der Übersichtlichkeit in mehrere Diagramme unterteilt. Bei einem Objekt, das mit dem Stereotyp <<Teilsystem>> gekennzeichnet ist, handelt es sich in Wirklichkeit um ein komplexeres Gebilde, das in einem weiteren Diagramm detailliert beschrieben wird. In den Sequenzdiagrammen taucht kSOAP als Teilsystem auf. Es zeigt die Zusammenarbeit des Vermittlerobjektes mit dem kSOAP-Framework. Am Ende dieses Unterkapitels wird dieses Teilsystem in einem Diagramm zusätzlich erläutert. Es kann allen Anwendungsfällen zu Grunde gelegt werden.

Das BootMidlet, das die Benutzungsschnittstelle implementiert, muss bei Vorgängen, die mindestens einen Netzwerkzugriff nach sich ziehen, einen nebenläufigen Prozess starten damit die GUI, trotz eines eventuellen Blockierens, weiter bedienbar bleibt. Der leichtgewichtige Prozess, auch *Thread* (in diesem Fall *LadeThread*) genannt, ist immer für einen einzigen solcher Vorgänge zuständig. Aus diesem Grund wird der LadeThread (s. z.B. Abb.5.5.1 und Abb.5.5.3) bei der clientseitigen Realisierung jedes Anwendungsfalles verwendet.

Die Benutzerangaben werden zu Beginn der Applikationslaufzeit eingesammelt, weil diese Daten allen vom Client initiierten Anwendungsfällen, außer dem Anwendungsfall *Adressangaben prüfen*, zugrunde liegen.

Sequenzdiagramme zum Anwendungsfall A. – Adressangaben prüfen

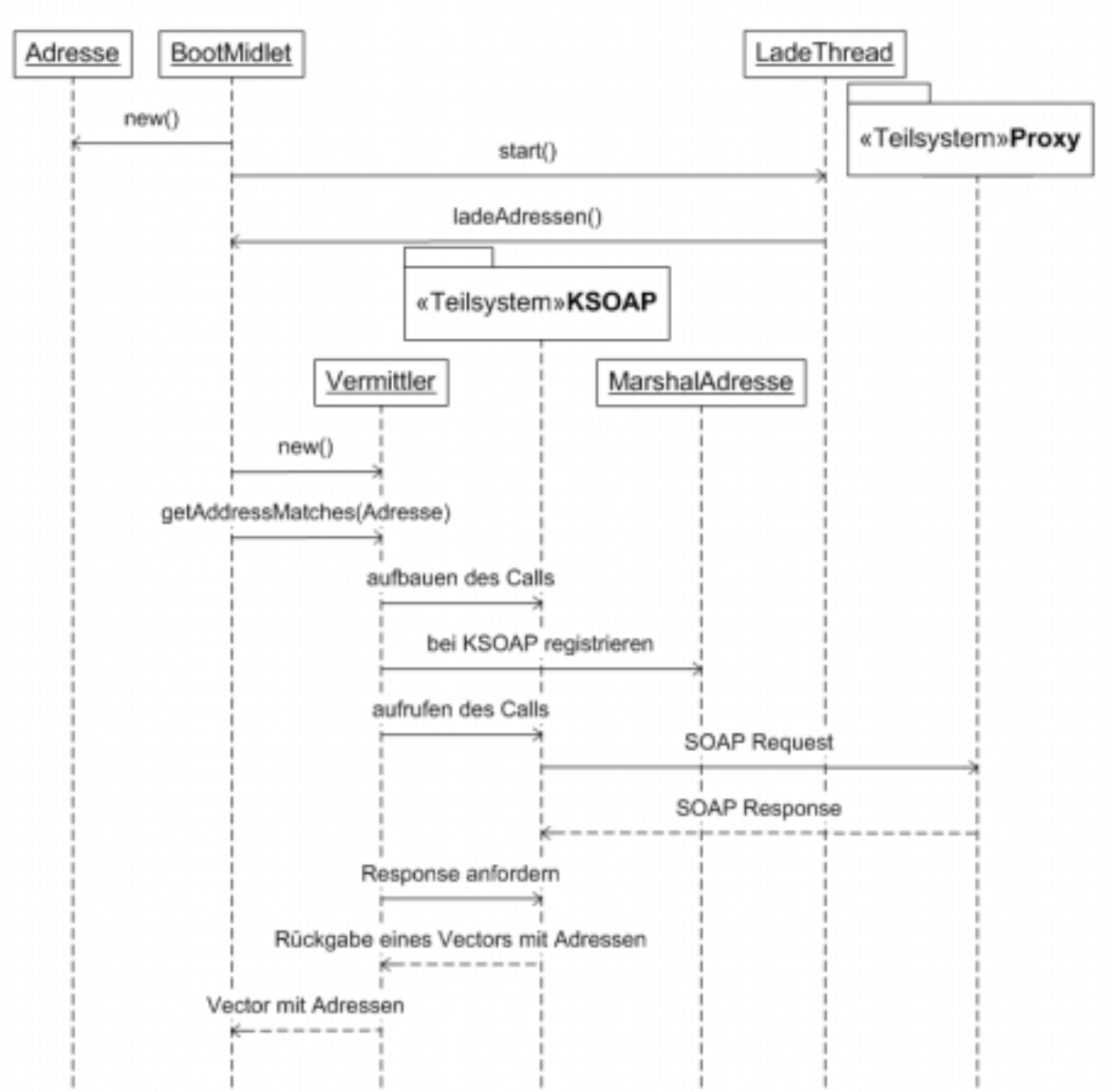

Abb. 5.5.1 Sequenzdiagramm – Prüfen der Adressangaben - clientseitig

Das Sequenzdiagramm in der Abbildung Abb.5.5.1 visualisiert das dynamische Verhalten der Objekte aus der Sicht des mobilen Clients. Die Adresse, die der Benutzer als den Ausgangspunkt angegeben hatte, muss zur Prüfung der Eindeutigkeit an den Proxy weitergegeben werden, damit dieser weitere Schritte bezüglich der Prüfung vornehmen kann. Dafür wird der Anwendungsfall *Adressen prüfen* in einem LadeThread vom BootMidlet aufgerufen. Dieser ruft wiederum eine der *lade*-Methoden des BootMidlets auf. In diesem Fall ist es die Methode *ladeAdressen*. Anschließend wird mithilfe der Methode *getAddressMatches(Adresse)* die Kontrolle über die Kommunikation zwischen dem mobilen Client und dem Proxy an das Vermittlerobjekt übergeben. Der Vermittler verwendet das kSOAP-Framework, um den entfernten Aufruf aufzubauen. Dabei werden zusätzlich spezielle De/Serialisierer-Klassen, die so genannten Marshaller, benutzt. Die eigentliche Kommunikation zwischen dem mobilen Client und dem Proxy-Server findet auf Basis von SOAP-Nachrichten statt. Es handelt sich um eine synchrone Kommunikation. Dabei wird eine SAOP-Anfrage gestellt und auf eine SOAP-Antwort gewartet. Der nebenläufige Prozess blockiert inzwischen.

Die weitere Verarbeitung der Adressenprüfung ist somit auf den Proxy verlagert worden, der in der Abbildung 5.5.1 als *Teilsystem Proxy* dargestellt wurde. Der weitere Verlauf kann der nächsten Abbildung Abb. 5.5.2 entnommen werden.

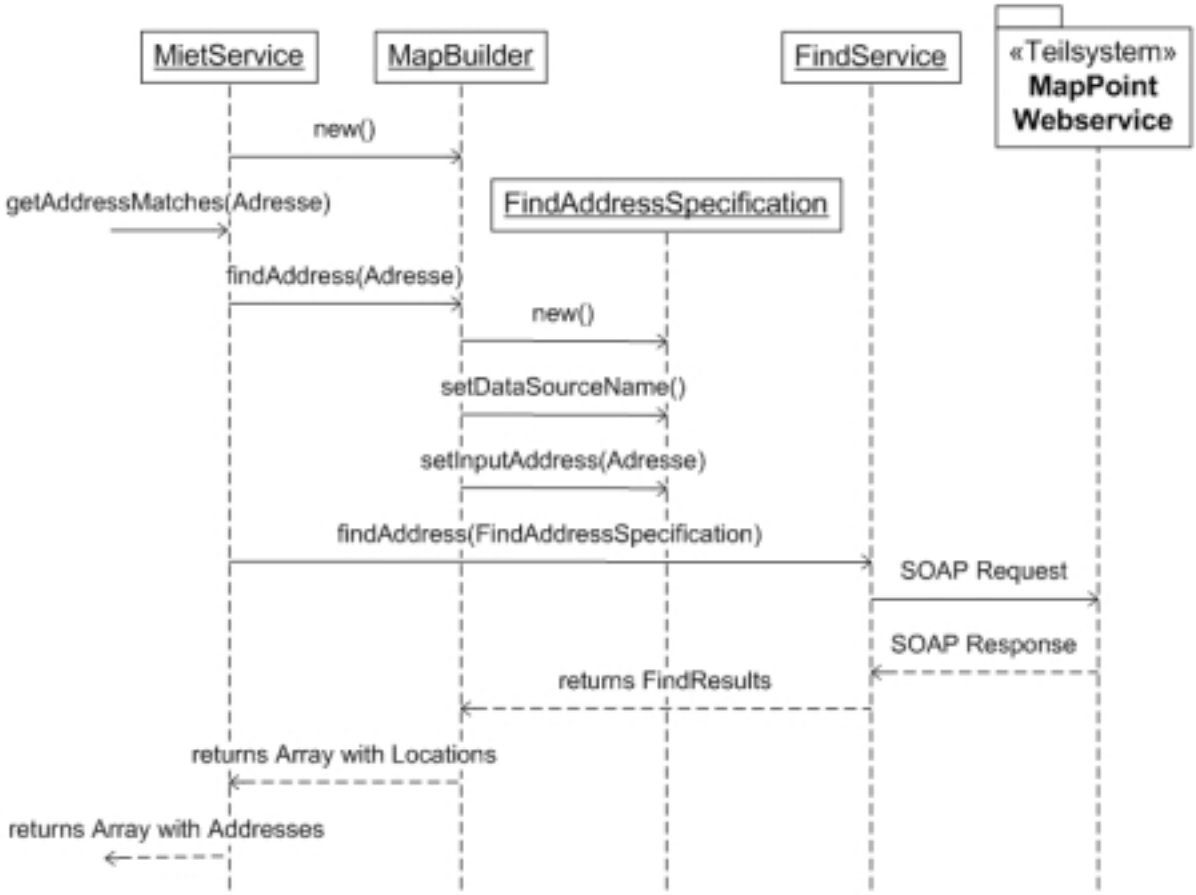

Abb. 5.5.2 Sequenzdiagramm – Prüfen der Adressangaben – serverseitig (Proxy)

Der MietService (s.Abb. 5.5.2) ist die Schnittstelle des Proxy-Servers, über die alle Anfragen mobiler Clients an den Proxy gestellt werden. Bei der Adressenprüfung wird die zu prüfende Adresse an den MietService vom mobilen Client weitergereicht. Die Prüfung der Adresse kann der Proxy jedoch nicht alleine bewältigen, dazu fehlen ihm die ortsbezogenen Informationen. Aus diesem Grund muss er die Adresse zur Prüfung an den MapPoint-Server leiten. Dieser wird die Prüfung vornehmen.
Als eine geeignete Schnittstelle des MapPoint-Servers für Suchvorgänge gibt es den Find-Service aus der MapPoint-Webservice-API, dessen Methode *findAddress()* die gewünschte Prüffunktionalität liefert.
Ähnlich wie der Vermittler des mobilen Clients die Kommunikation zw. dem Client und dem Proxy sicherstellt, sorgt der MapBuilder des Proxy für die Kommunikation zwischen dem Proxy und dem MapPoint-Server. Für den entfernten Aufruf baut der MapBuilder eine Adressspezifikation auf. Dabei wird neben der Adressangabe, die Datenbank einer Region spezifiziert z.B. *MapPoint.EU* für Europa. Diese Datenbank muss für eine Adresssuche die Fähigkeit *CanFindPlaces* besitzen, was hier auch der Fall ist.
Bei dem Find-Service-Objekt handelt es sich um den Stub, der sich auf dem Proxy befindet.
Zwischen dem Proxy und dem MapPoint-Server findet der Nachrichtenaustausch ebenfalls über SOAP statt.
Bei einer erfolgreichen Prüfung der Adresse ist das Ergebnis, das der Mietservice an den Client zurückschicken kann, ein Array mit einer einzigen Adresse. Im Falle einer nicht

eindeutigen Adresse wird das Array mehrere Adressalternativen enthalten. Falls die Adresse ganz falsch sein sollte, wird eine *NoSuchAddressException* geworfen, die auf dem mobilen Client entsprechend aufbereitet wird.

Sequenzdiagramme zum Anwendungsfall B. – Boot suchen

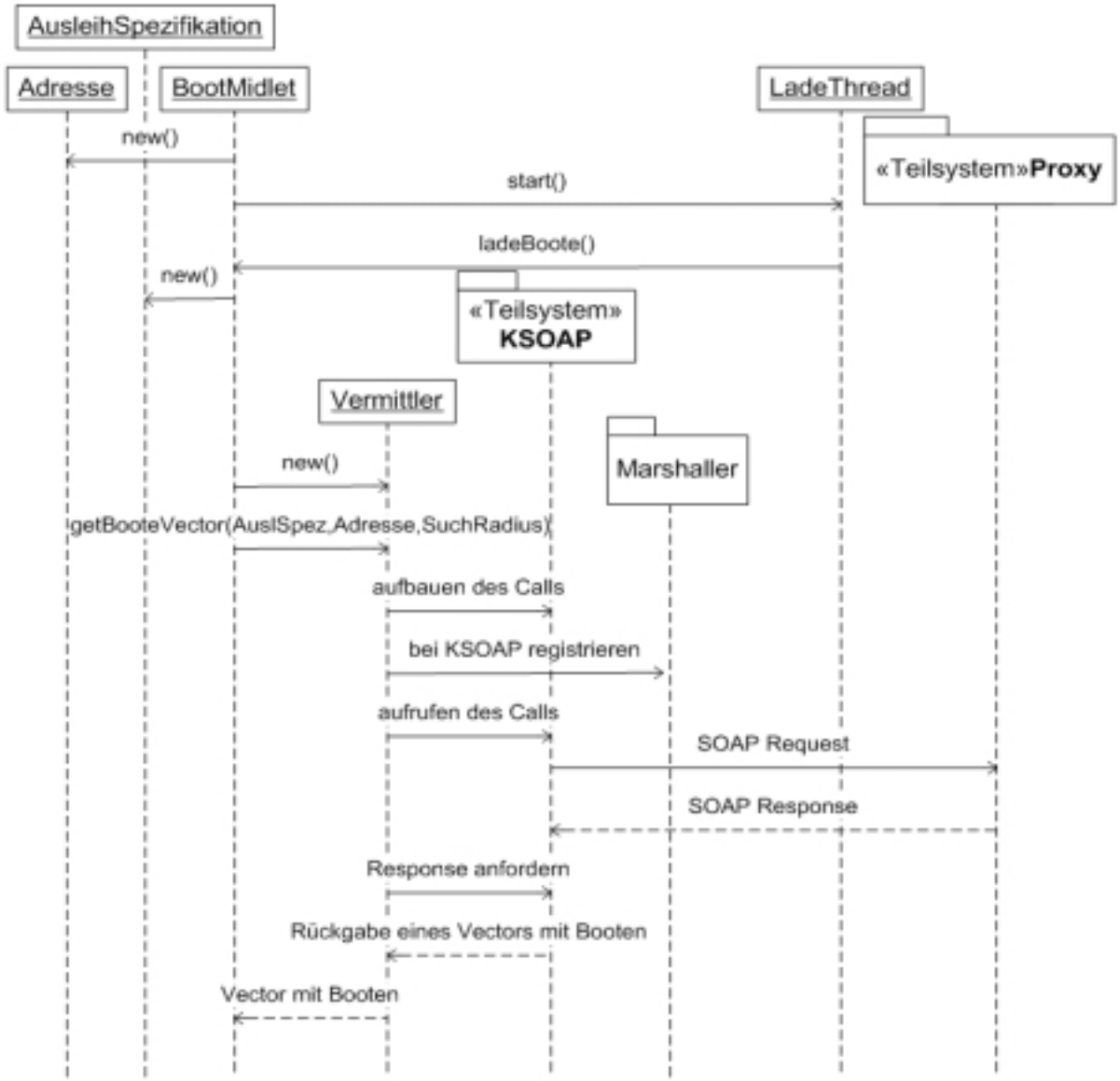

Abb. 5.5.3 Sequenzdiagramm – Boot suchen – clientseitig

Das Sequenzdiagramm in der Abbildung Abb.5.5.3 ähnelt sehr dem Diagramm 5.5.1, weil es sich ebenfalls um den clientseitigen Aufruf einer entfernten Methode des Proxy-Servers handelt. Das Adressenobjekt repräsentiert auch hier den Ausgangspunkt der Suche, mit dem Unterschied jedoch, dass diese Adresse bereits geprüft wurde. Der Anwendungsfall *Boot suchen* wird auch hier in einem *LadeThread* vom *BootMidlet* aufgerufen. Dieser ruft die Methode *ladeBoote()* des Midlets auf. Anschließend wird mithilfe der Methode *getBooteVector()* die Kontrolle über die Kommunikation zwischen dem mobilen Client und dem Proxy an das Vermittlerobjekt übergeben. Dabei wird die *AusleihSpezifikation*, die *Adresse* und der Suchradius als Parameter übergeben. Der Vermittler verwendet das kSOAP-Framework um den entfernten Aufruf aufzubauen, dabei werden die Marshaller für die De/Serialisierung verwendet. In der Abbildung 5.5.3 sind mehrere *Marshaller*, zu Gunsten der Übersichtlichkeit, in einem Paket zusammengefasst worden. Dabei handelt es sich um den

MarshalBoot, den *MarshalProvider*, den *MarshalAdresse*, den *MarshalAusleihSpezifikation* und um den *MarshalDate*, der bereits im kSOAP-Framework enthalten war. Auch hier findet die Kommunikation des mobilen Clients mit dem Proxy-Server auf Basis von SOAP-Nachrichten statt.
Die weitere Verarbeitung der Bootsuche wird auf dem Proxy fortgesetzt. Der Proxy wird in der Abbildung 5.5.3 als ein *Teilsystem* dargestellt. Der weitere Verlauf kann der nächsten Abbildung (Abb. 5.5.4) entnommen werden.

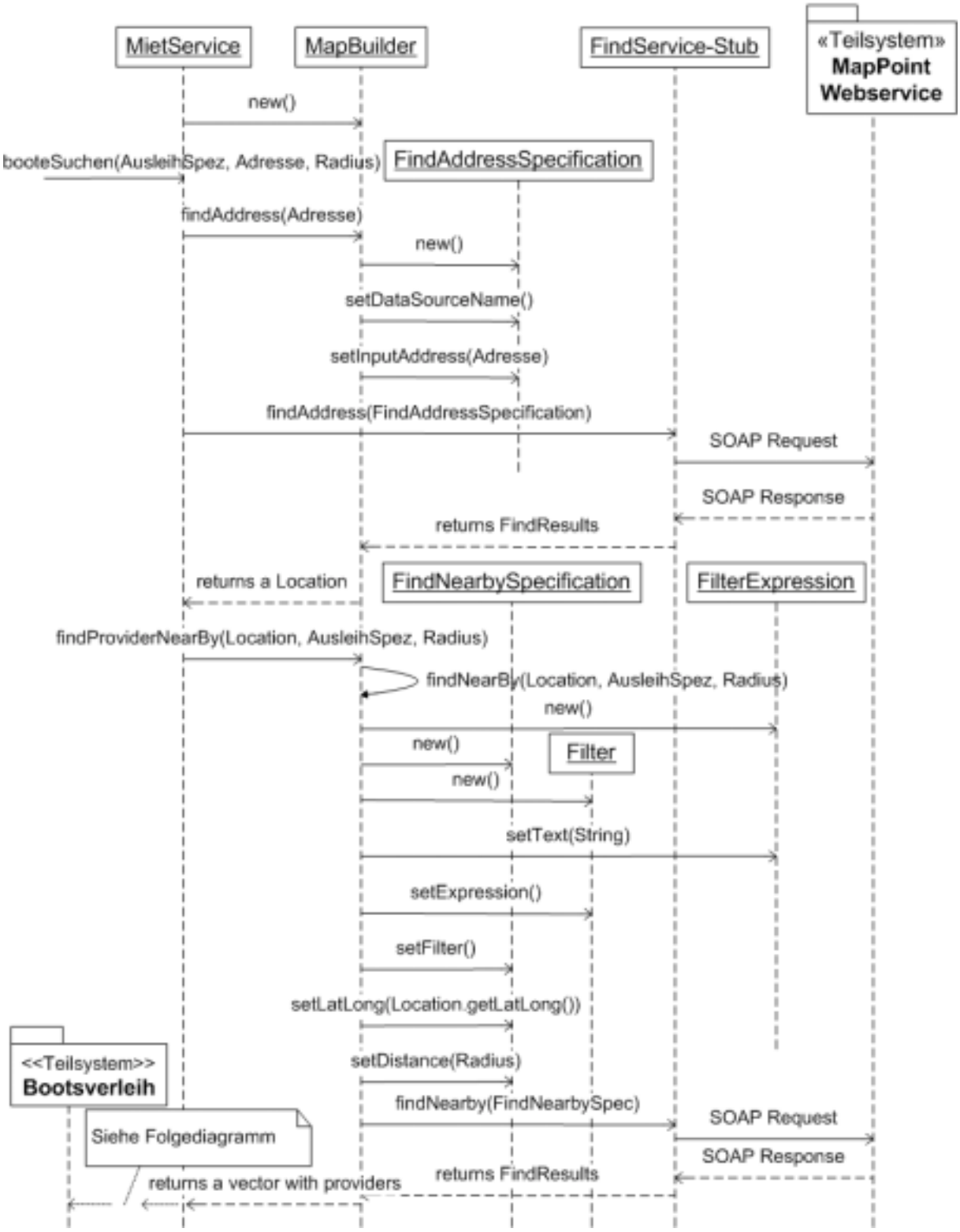

Abb. 5.5.4 Sequenzdiagramm – Boot suchen – serverseitig (Proxy)

Die serverseitige Implementierung des Anwendungsfalles *Boot suchen* ist so umfangreich, dass sie in zwei Sequenzdiagrammen dargestellt werden musste. So ist die Abbildung 5.5.4 der erste Teil und die Abbildung 5.5.5 der zweite Teil dieses Anwendungsfalles. Die Komplexität dieses Anwendungsfalles liegt darin, dass drei entfernte Webserviceaufrufe aufgebaut und durchführt werden müssen. Die ersten beiden Aufrufe betreffen den MapPoint-Webservice und sind in der oberen Grafik (Abb. 5.5.4) zu sehen. Dabei gilt der erste Aufruf, der Umwandlung der benutzerdefinierten Ausgangsadresse in eine Location. Eine Location ist eine von MapPoint geprüfte und geokodierte Adresse, die mit Sicherheit existiert. Beim zweiten MapPoint-Aufruf gilt es, die Bootsverleiher (Provider) der näheren Umgebung des Nutzers herauszufinden. Dazu wird die Methode *findNearBy()* des FindServices aus der MapPoint-Webservice API verwendet. Die Methode liefert eine MapPoint-spezifische Collection mit FindResult-Objekten, die den Providerobjekten der von uns definierten MapPoint-Datenbank. Der MapBuilder wandelt diese Entities in Provider-Objekte um und gibt sie, in einem Vector verpackt, an den MietService zurück. An dieser Stelle sollten jedoch nicht alle Provider der Umgebung herausgesucht werden, sondern nur die, welche zusätzlich die geeignete Antriebsart von Booten anbieten. Dies wird durch einen Filterausdruck(*FilterExpression*) erreicht, der innerhalb eines Filters an die *FindNearbySpecification* übergeben wird. Die Spezifikation wir dann, mithilfe der *findNearBy*-Methode des *FindService-Stubs* an den MapPoint-Server geschickt. Der Text der *FilterExpression* erhält einen String mit einem SQL ähnlichen Ausdruck, der wie folgt aufgebaut ist: *SupportedModeOfDrive=AusleihSpez.getAntriebsart()*. *SupportedModeOfDrive* ist der Attributname für die Antriebsart, die der Bootsverleih anbietet. Dieser Name wird mit der, vom Benutzer gewünschten, Antriebsart verglichen. Der Vergleich wird durch den Gleichheitsoperator erzwungen. Die benutzergewollte Antriebsart wird der *AusleihSpezifikation* entnommen.
Nun, da die richtigen Provider ermittelt wurden, kann die Suche nach Booten auf den Provider-Servern vorgenommen werden. Dies umfasst den dritten Webserviceaufruf des Anwendungsfalles *Boot suchen* und ist im folgenden Sequenzdiagramm (Abb. 5.5.5) visualisiert worden.

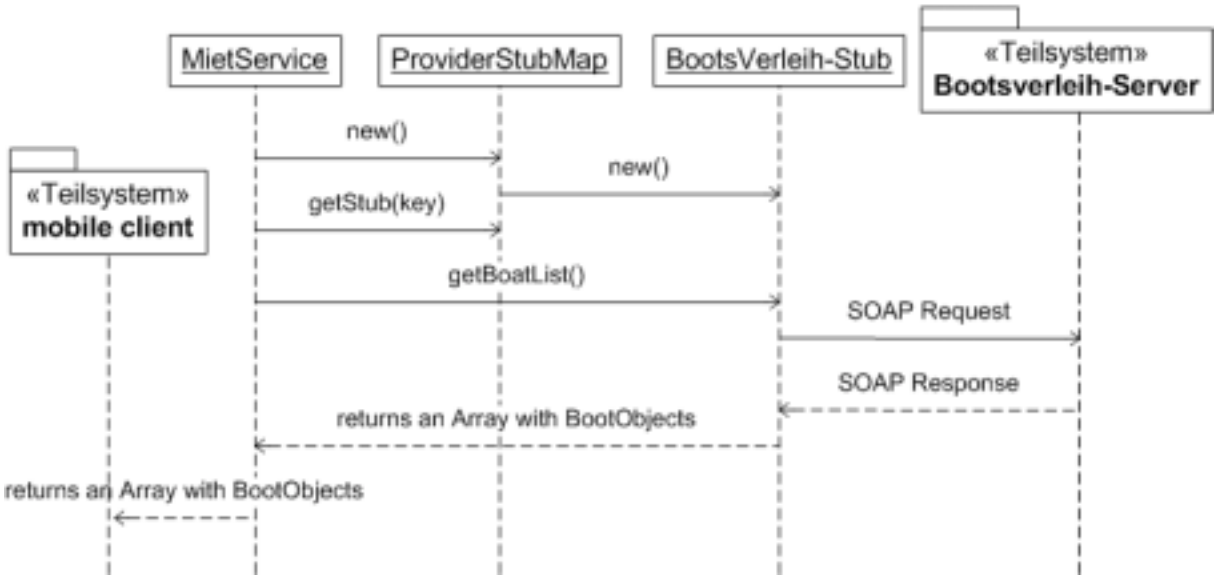

Abb. 5.5.5 Sequenzdiagramm – Boot suchen – serverseitig (Proxy) - Fortsetzung

Um dem mobilen Client die gewünschten Boote liefern zu können, muss der MietService bei allen vorher ermittelten Bootsverleihern (s. Abb. 5.5.4) die getBoatList-Methode aufrufen. Das bedeutet, dass der im Diagramm beschriebene Algorithmus bei mehr als einem

ermittelten Provider, mehrmals wiederholt werden muss. Für den entfernten Aufruf beim Bootsverleih-Server muss ein entsprechender Stub erzeugt und in einer ProviderStubMap abgelegt werden. Sowohl die Stuberzeugung als auch die Pflege der StubMap sind Aufgaben des Systemadministrators. Sobald die Stubs in der StubMap vorliegen, kann sich der MietService mithilfe der Methode getStub() bedienen. Der Stub des jeweiligen Providers wird anhand des Providerschlüssels ermittelt.
Letztendlich erhält der mobile Client ein Array mit Bootobjekten, das bei dem clientseitegen Marshalling durch kSOAP in einen Vector mit Bootobjekten umgewandelt wird.
Sowohl der mobile Client als auch der Bootsprovider-Server sind im Diagramm als Teilsysteme dargestellt worden, denn es handelt sich dabei um komplexere Infrastrukturen und nicht um Objekte von Klassen.

Sequenzdiagramm zur Kommunikation über kSOAP

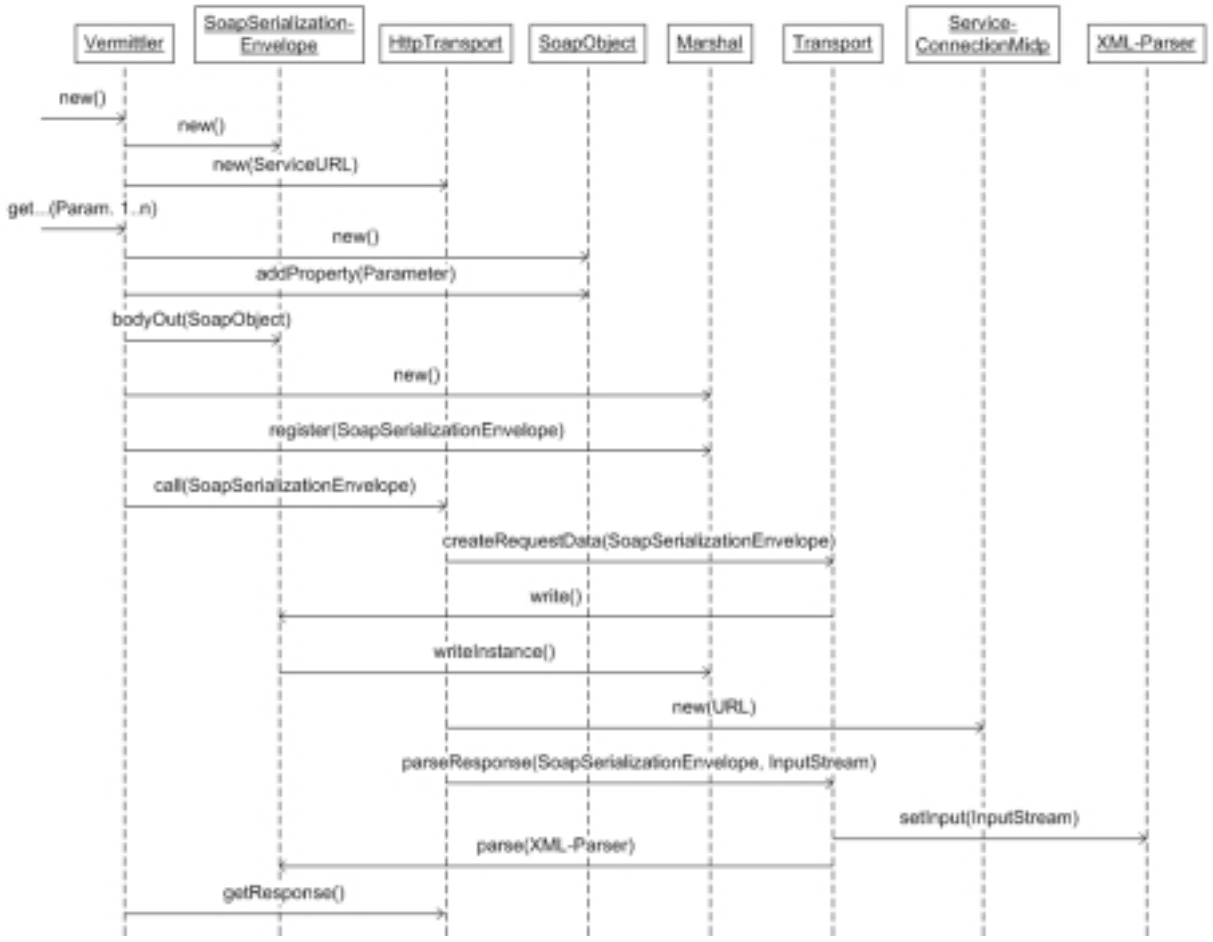

Abb. 5.5.6 Sequenzdiagramm – Kommunikation in kSOAP beim Serialisieren

Das Diagramm in der Abbildung 5.5.6 verdeutlicht die Zusammenarbeit zwischen der Vermittlerkomponente des mobilen Clients mit dessen kSOAP-Webservice-Framework beim Abschicken eines Objektes über SOAP. Das Marshallobjekt steht hier stellvertretend für mehrere Marshallobjekte, die für alle komplexen Objekte des Clients vom Cliententwickler implementiert werden müssen. Bei vielen Webserviceframeworks gehören die Funktionalitäten solcher Marshallerobjekte zum Framework dazu. Im Falle von kSOAP wird die Methode writeInstance() implementiert um ein Objekt in den SOAP-Strom zu integrieren. Die readInstance()-Methode dient dem Herauslesen von Objekten aus dem SOAP-Strom.

Letztendlich wird der Marshaller mithilfe der register()-Methode bei einer SoapSerializationEnvelope registriert.
Mit diesem Sequenzdiagramm wird das Designkapitel geschlossen, und nun soll noch mal das was in dem Kapitel besprochen wurde, zusammengefasst werden.

5.6 Resümee

In der Designphase der Entwicklung wurde, auf der Grundlage der vorangegangenen Analyse, das *mobile ortsbasierte Bootsverleihsystem* mit dessen Komponenten entworfen. Der Softwareentwurf ist in dem vorliegenden Designkapitel beschrieben worden. In dieser Entwicklungsphase mussten wichtige Designentscheidung getroffen werden, wobei die wichtigsten im Kap. 5.1 aufgeführt wurden. Funktionale Komponenten wurden auf die Systemarchitektur verteilt, damit die Verantwortlichkeiten der Systemeinheiten, wie der mobile Client, der Proxy-Server und die externen Server, sichtbar werden. Die Verteilung findet man im Kap. 5.2 . Die Brücke zwischen dem funktionalen Analysemodell und der technischen Implementierung bilden die, im Kap. 5.3 eingeführten, technischen Komponenten. Im Kap. 5.4 wurde kurz darauf eingegangen, wie sich die Sicherheit und Robustheit des Systems sicherstellen lassen. Das dynamische Verhalten der Objekte des Systems wurde anhand von Sequenzdiagrammen erläutert.

Das Designkapitel ist eines der umfangreichsten Kapitel dieser Diplomarbeit geworden. Bei der Erklärung der Sequenzdiagramme sind bereits wichtige Implementierungsdetails berührt worden. Damit wird das Implementierungskapitel relativ kurz gehalten und nimmt vor allem Bezug auf die verwendeten Werkzeuge.

Kapitel 6. Implementierung

In diesem Kapitel werden Werkzeuge, Laufzeitumgebungen und Entwicklungsumgebungen aufgeführt, die zur Realisierung des Prototypen verwendet wurden (s.Kap 6.1). Außerdem wird noch mal konkret auf die Automatisierung des Registrierungsvorgangs eingegangen (s.Kap.6.2). Schwierigkeiten, die während der Arbeit entstanden sind, werden im Kapitel 6.3 aufgeführt.

6.1 Verwendete Werkzeuge

Laufzeitumgebung des zentralen Rechners
Als Laufzeitumgebung des Entwicklungsrechners, der sowohl als Entwicklungsmaschine als auch als Proxyserver der Anwendung gedient hatte, wurde das Java(TM) 2 SDK, Standard Edition in der Version 1.4.2 eingesetzt.

Laufzeitumgebung des mobilen Clients
Java 2 Micro Edition ist die Laufzeitumgebung für den Client. Die J2ME-Bibliothek wird aus mehreren Bibliotheken zusammengebaut. Dabei handelt es sich um die verschiedenen Konfigurationsbibliotheken wie *CLDC*, Profilbibliotheken wie *MIDP* und Bibliotheken optionaler Pakete wie z.B. das *jsr082* für die Unterstützung von Bluetooth. Für die Entwicklung des Prototypen wurden erstmals nur Bibliotheken für *CLDC* und *MIDP* verwendet. Die gesamte J2ME-Bibliothek ist Bestandteil des *J2ME Wireless Toolkits*.

J2ME Wireless Toolkit 2.2
Das Toolkit beinhaltet APIs für *J2ME*, unterstützt das *MIDP* in der Version 2.0 und bietet einige Emulationsprogramme mobiler Geräte. Eines dieser Programme wird zur Präsentation des Prototypen verwendet. Das Toolkit musste in die Entwicklungsumgebung integriert werden.

Die Entwicklungsumgebung
Eclipse SDK, das in der Version 3.1.2 verwendet wurde, hat sich als ein sehr komfortables und zuverlässiges Werkzeug sowohl für die Entwicklung von Webservices als auch für die Entwicklung von Applikationen für mobile Geräte.

Verwendete *Plugins*, die zusätzlich zum Standardumfang von *Eclipse* installiert werden mussten sind:

EclipseME Version 1.5.0 – Dieses Plugin ermöglicht die Integration einer J2ME-Anwendung in die Entwicklungsumgebung.

SYSDEO Tomcat Plugin in der Version 3.1.0: Erleichtert den Umgang mit dem Tomcat-Webserver, der in Wirklichkeit eher ein Servletcontainer ist. Er eignet sich sehr gut für den Einsatz des Axis-Frameworks, welches ein Servlet basiertes Webserviceframework ist.

Das Webservice-Framework für den mobilen Client - kSOAP
In der Arbeit wurde das open-source-Produkt kSOAP2 in der Version 2.1.0 beta verwendet. Dieses Framework hat die Entwicklung des Webservice-Clients auf einem mobilen Gerät ermöglicht.

Das Webservice-Framework für den Proxy - Axis
Auf dem Proxy-Server wurde sowohl für die clientseitige als auch für die serverseitige Webserviceimplementierung das Axis-Framework eingesetzt. Zum Einsatz kam eine Mischung der Version 1.0 RC1 und der Version 1.2. Diese Mischung ist auf den Einsatz einer älteren Version der *MapPoint-Webservice API* für Java, zurück zu führen. Axis bietet Werkzeuge, die aus einer WSDL-Datei Javacode erzeugen können und auch umgekehrt.

Die MapPoint-Webservice API
Die Klassen der MapPoint-Webservice API (Version 3-0) wurden aus einer WSDL-Datei auf dem zentralen Server mithilfe von Axis generiert. Damit war der MapPoint-Webservice über seine Stubs erreichbar. Der CustomerDataService

6.2 Vereinfachung des Registrierungsvorgangs durch Automatisierung

Für den Betrieb des zentralen Servers des Verleihsystems wird der Einsatz eines Administrators vorausgesetzt. Die Gründe dafür wurden bereits bei den Designüberlegungen (s. Kap. 5-Design) angegeben. Der Administrator erstellt Stubs, wobei jeder Stub einem einzelnen Bootsverleih-Webservice entspricht. Das ist relativ umständlich, weshalb während der Systementwicklung überlegt wurde, wie eine Vereinfachung dieses Vorgehens ermöglicht werden könnte. Zur Lösung des Problems müsste eine abstrakte Schnittstelle entwickelt werden, die allen Stubs gemeinsam wäre.

Auf der Ebene der Programmiersprachen, wie z.B. Java, löst man solche Angelegenheiten entweder mit dem Einsatz einer abstrakten Oberklasse von der die konkreten Unterklassen alle Methoden erben oder eines gemeinsamen Java-Interfaces, das die Klassen implementieren. Damit hätten die entfernten Clients (d.h. die Repräsentanten der Provider Webservices), eine gemeinsame Schnittstelle, über die man zur Übersetzungszeit (d.h. zur Programmentwicklungszeit) alle Provider auf die gleiche weise ansprechen kann. Dafür wird jeder Providerservice das Interface *BootsVerleih* implementieren. Das wird entsprechend auch bei dem entfernten Client (in diesem Fall auf dem zentralen Server) bei der Generierung der clientseitigen Stubklassen berücksichtigt.

Da die Schnittstellen der Provider über WSDL-Dokumente beschrieben werden, muss zunächst eine gemeinsame WSDL aller Provider erzeugt werden, die sich dann auf dem zentralen Server befinden wird. Dafür wird zunächst der abstrakte Teil der gemeinsamen WSDL (PortTypes, Bindings usw.) generiert. Auf den Servern der Provider wird jeweils der konkrete Teil der Provider-WSDL (Services und Ports) erzeugt. Die konkreten Teile sollen zu der abstrakten WSDL hinzugefügt werden. Zur Zeit geschieht das assemblieren von Hand, da es sich aber um XML-Konstrukte handelt, ist das Automatisieren möglich.
Beim Zusammensetzen des gemeinsamen WSDL-Dokumentes ist zu beachten, dass die Provider, ein gemeinsames Serviceelement haben müssen, und lediglich durch Ports unterscheidbar sein sollen. Aus dieser WSDL müssen mit dem Axis-Tool *wsdl2java* die Java-Dateien ins provider.client-Package generiert werden.
Die Provider müssen zusätzlich zu der WSDL mit der Implementierung d.h. dem konkreten Teil der WSDL auch noch eine zusätzliche WSDL erstellen, die alles enthält d.h. auch den abstrakten Teil, um die WSDDs (fürs Un/Deployment) erzeugen zu können.

Die hier besprochene Vereinfachung des Registrierungsvorgangs wurde beim Prototypen umgesetzt, wobei nicht alle erwarteten Vorteile herausgeholt werden konnten. Das aufgrund

einiger Schwierigkeiten, die weiter unten (unter „Probleme mit multiplen Ports“) beschrieben werden.

6.3 Welche Schwierigkeiten sind aufgetreten?

Probleme mit Axis bei mehreren Kontextpfaden (zB. Axis, Bootgigant, Bootsbauer und Sportboote):
Beim Deployen, trägt Axis die Services aus Providerprojekten in die *Server-Config.wsdd* des axis-Projektes ein. Diese mussten anschließend von Hand in die geeignete *Server-Config* kopiert werden. Dieses Problem wird sich jedoch erübrigen, sobald die Projekte auf verschiedene Rechner portiert werden, denn dann kann der gleich Kontextname für jeden Bootsverleihservice verwendet werden ohne mit anderen zu kolidieren.

Probleme mit multiplen Ports:
Beim Versuch die einzelnen Providerservices über eine gemeinsame abstrakte Schnittstelle anzusprechen schien die Idee zunächst umsetzbar zu sein. Mit Methoden des *BootsVerleihServiceLocator* kann man die einzelnen Providerimplementierungen herausbekommen. Dafür stehen die *getPort()* Methoden zur Verfügung, die jeweils ein Remoteobjekt liefern. Da für dieses Projekt die automatische Wahl der Stubs eine wichtige Rolle spielt, wäre an dieser Stelle eine Iteration über die Ports sinnvoll. Bis hierher ist die Idee mit den Ports fast aufgegangen. Die eben erwähnte Locatorklasse besitzt sogar eine Methode, die einen Iterator für die qualifizierten Namen der Ports zurückgibt.
Axis macht somit den Anschein diese Idee der multiplen Ports zu unterstützen, dem ist jedoch leider nicht so. Dies ist z.B. dem generierten Kommentar der Methode *getPort* zu entnehmen.
“This service has multiple ports for a given interface,
the proxy implementation returned may be indeterminate.”
Es wird hier auf ein nicht deterministisches Verhalten der Methode getPort hingewiesen, falls man mehrere Ports verwenden wollte. Im Klartext heißt das, dass nur der zuerst gefundene Port als ein solcher berücksichtigt wird. Der Portiterator kennt auch nur den einen Port und ist deshalb unbrauchbar.
Die Ansätze für multiple Ports sind in Axis gegeben, jedoch nicht vollständig Implementiert.

6.4 Fazit
Das Implementierungskapitel ist relativ kurz geworden, jedoch liefert es wichtige Details für den Aufbau einer geeigneten Implementierungsumgebung, da in dieser Arbeit bewährte Werkzeuge und Systeme mit Angabe der Versionsnummern aufgeführt wurden. Außerdem sollte in diesem Kapitel auf einige Probleme der Implementierungsphase und vor allem das Problem der Automatisierung hingewiesen werden. Dies waren die relevantesten Schwierigkeiten. Natürlich gab es noch weitere Tücken, die jedoch aus Zeitgründen nicht aufgeführt werden konnten.

Die relevanten Ergebnisse der Diplomarbeit wurden somit in dieser Ausarbeitung erfasst. Die Arbeit kann mit dem nächsten Kapitel zusammengefasst und abgeschlossen werden.

Kapitel 7. Fazit und Ausblick

Die vorliegende Arbeit beschreibt die Durchführung einer Machbarkeitsstudie und den daraus abgeleiteten Entwurf eines ortsbasierten Webservice für einen mobilen Client auf Basis vom Java-MIDP kSOAP. Das kSOAP-Framework hat sich, trotz einiger Einschränkungen, als ein gutes Werkzeug für die Entwicklung mobiler Webserviceclients herausgestellt. Als geeignete Architektur für den smart Client, wurde aus Gründen der Ressourcenbeschränkung eine serverseitige Proxyarchitektur gewählt.
Die Machbarkeitsstudie sollte zeigen, ob ein ortsbezogener Verzeichnisdienst für Webservices realisierbar ist. Dafür sind bekannte geographische Informationssysteme auf ihre Eignung untersucht worden. Dabei wurde MapPoint als das am besten geeignete System gewählt. Die Arbeit hat gezeigt, dass MapPoint eine mit Einschränkungen geeignete Infrastruktur für die Implementierung einer ortsbasierten Registry für Webservices ist. Die Suche nach Bootsverleihdiensten mit einer kartenbasierten Anzeige, die Suche nach Booten der näheren Umgebung des Standorts eines Nutzers, als auch deren Reservierung ist möglich. Auch ein kartenbasierter Routenplaner konnte realisiert werden. Die Machbarkeit des gewählten Ansatzes ist in einer durchgängigen Implementierung vom mobilen Client über den Proxy bis hin zu den Businessservern, darunter auch der MapPoint Server, nachgewiesen worden. Dabei wurden die Beschränkungen der verwendeten Software, insbesondere die von MapPoint und kSOAP, für die Lösung der Aufgabenstellung herausgearbeitet. Das größte Problem von MapPoint hinsichtlich der Verwendung für den Bau einer Registry ist, die fehlende Funktionalität zum Ein- und Austragen einzelner Datensätze der MapPoint-Datenbank. Stattdessen können diese Vorgänge nur über die Download- und Upload-Jobs des Customer Data Services umgesetzt werden. Zum Einen sind diese Vorgänge dadurch in ihrer Anzahl begrenzt, zum zweiten sind sie nicht sehr performant. Solange dieses Problem seitens der Firma Microsoft nicht behoben wird, können höchstens 25 Bootsverleiher am Tag registriert werden.
Das Beherrschen des kSOAP-Frameworks bedarf einer längeren Einarbeitungsphase. Dies vor allem aufgrund mangelnder Dokumentation. Da kSOAP ein Open-Source-Produkt ist, konnte die Funktionsweise nur mit viel Arbeitsaufwand erkundet werden. Weiterhin kann an kSOAP bemängelt werden, dass sich der Entwickler um technische Details, wie das Marshalling, selber kümmern muss.

Insgesamt können der Systementwurf und der implementierte Prototyp als gelungen bezeichnet werden. Die wesentlichen Funktionalitäten konnten im Rahmen der Arbeit realisiert und die wichtigsten Anwendungsfälle abgedeckt werden. Aufgrund der modularen Entwicklung der Software stellt es keinerlei Probleme dar, diese beliebig weiterzuentwickeln.

Bei einer Erweiterung der bestehenden Architektur um den direkten Zugriff aufs Internet über GSM/GPRS oder UMTS, könnten weitere Eigenschaften mobiler Systeme genutzt werden. Mithilfe von Push-Services z.B., kann der Benutzer des Systems, dank seiner uneingeschränkten Erreichbarkeit mit personalisierten d.h. auf sein Profil zugeschnittenen, Nachrichten versorgt werden. Vorausgesetzt, das Clientprogramm hat dem Serverprogramm seine IP - Adresse mitgeteilt und läuft zumindest im Hintergrund. Wenn man den Client um einen GPS-Empfänger erweitern würde, wäre er lokalisierbar und eine derartige Nachricht könnte eine Liste der in der Umgebung befindlichen Yachtklubs bzw. Bootsverleihstellen enthalten.

Hinsichtlich der Buchung von Booten, beschränkt sich der Prototyp auf die Reservierung von Booten. D.h. dass die Bezahlung softwaretechnisch nicht geregelt ist. Die endgültige Buchung von Booten könnte über einen Buchungsservice geschehen. Der Buchungsservice (im engl.

ticketing service), wäre als eine mobile Anwendung im Bereich Mobile Commerce einzuordnen. Diese Klasse von Anwendungen umfasst das Bankwesen, den Handel, Einkauf, Auktionen und andere.

Mit diesen Erweiterungsvorschlägen, möchte ich diese Diplomarbeit abschließen und mich bei Ihnen für Ihr Interesse, diese Arbeit betreffend, bedanken.

Kapitel 8. Literaturhinweise

Literatur zu GIS

[Burrough1988] Peter A. Burrough and Rachel A. McDonnell, *Principles of Geographical Information Systems*, Oxford University Press, 1988

[DeMers1997] Michael N. DeMers, *Fundamentals of Geographic Information Systems*, John Wiley & Sons, 1997

[BillGeo] Ralf Bill, *Grundlagen der Geo-Informationssysteme Band 1 Hardware, Software und Daten*, Herbert Wichmann Verlag, 1999, Kap. 1, 3, 7

[SNA] National Atlas of Sweden, URL - http://www.sna.se

[WebMap] Asche/Herrmann, *Web.Mapping 2 Telekartographie, Geovisualisierung und mobile Geodienste*, Herbert Wichmann Verlag, Apr. 2003

[Zorg] Gesundheitsatlas der Niederlande, URL - http://www.zorgatlas.nl

[Dickmann2001] Frank Dickmann, *Compass Das Geographische Seminar, web-mapping und web-gis*, Westermann Schulbuchverlag, 2001

[OGC] OGC-Spezifikationen, URL - http://www.opengis.org/techno/specs.htm

Literatur zu Webservices

[Minouru2005] Dr. Minouru Etoh, *Next Generation Mobile Systems 3G and Beyond (Kapitel 9)*, John Wiley & Sons, 2005

[Eberhart2003] Andreas Eberhart, Stefan Fischer, *Webservices Grundlagen und praktische Umsetzung mit J2EE und .NET*, Carl Hanser Verlag, 2003

Literatur zu Mobilen Geräten

[Handhelds] Handhelds.org, URL - http://www.handhelds.org/

[Wiehler2004] Gerhard Wiehler, *Mobility, Security und Webservices – Neue Technologien und Service-orientierte Architekturen für zukunftsweisende IT-Lösungen* , Herausgeber: Siemens, Verlag: Publicis Corporate Publishing, Erlangen 2004

Literatur zu Java auf mobilen Geräten

[Qusay2002] Qusay H. Mahmoud, *Learning Wireless Java*, O'Reilly, 2002

[JavIBM] *IBM WebSphere Studio Device Developer*, URL - http://www-3.ibm.com/software/pervasive/products/wsdd

[JavSava] Homepage von SavaJe, URL - http://www.savaje.com/

Andere Literatur und Links

[Gamma2001] Erich Gamma, Richard Helm, Ralph Johnson, John Vlissides, Entwurfsmuster-Elemente wieder verwendbarer objektorientierter Software, Addison Wesley Verlag, 1996 (korrigiert 2001)

[MapPointWS] *Der Mappoint Webservice*, URL - http://www.mappoint.msn.com

[MapPointDB] *MapPointWebserviceDataSources*, URL - http://msdn.microsoft.com/library/default.asp?url=/library/en-us/mappointsdk/html/index.asp

[MapPointJava] *MapPointWebservice mit Java (samples)*, URL - http://go.mappoint.net/mappointmpac/

[Mapdex] *MAPDEX – eine Spezialsuchmaschine für WEB GEO SERVICES* (Suchbegriffe zum Suchen nach allen Webservices: wms;ims;wfs usw), URL - http://www.mapdex.org

[ApacheAxis] Apache Axis, URL - http://ws.apache.org/axis/

[ApacheSoap] Apache Soap, URL - http://ws.apache.org/soap/

[WSPython] Webservices mit Python, URL - http://pywebsvcs.sourceforge.net/

[JSR172] JSR 172: J2ME Webservices Specification, URL - http://jcp.org/jsr/detail?id=172

[kSOAP] kSOAP, URL - http://kobjects.org/auto?self=$81d91ea1000000f5b22b3fc4

[SOA1] URL - http://www.sdm.de/web4archiv/objects/download/fachartikel/sdm_soa_nutzen_richter.pdf , betrachtet am 18.01.2006

[W3C a] *SOAP Version 1*.2. 2004. – URL http://www.w3.org/TR/2003/REC-soap12-part0-20030624/

[W3C b] *W3C Webservices: Architektu*r – URL http://www.w3.org/TR/2003/WD-ws-arch-20030514/

[W3C c] *W3C Webservices: Glossa*r – URL http://www.w3.org/TR/2003/WD-ws-gloss-20030514/

[WS-Reliability] WS-Reliability Spezifikation von OASIS – URL http://docs.oasis-open.org/wsrm/ws-reliability/v1.1/wsrm-ws_reliability-1.1-spec-os.pdf

[UserLand] XML-RPC, von UserLand Software - URL http://www.xmlrpc.com/

[Middlewarebeschreibung von SEI] Middlewarebeschreibung vom Software Engineering Institute, URL - http://www.sei.cmu.edu/str/descriptions/middleware.html

[MTS] Microsoft Transaction Server – URL
http://www.microsoft.com/germany/msdn/library/servers/MicrosoftTransactionServerInAktion.mspx

Kapitel 9. Glossar

DOM (Document Object Model) – ist eine vom W3C definierte Programmierschnittstelle, die dynamischen Zugriff auf XML bzw. HTML – Dokumente ermöglicht. Die Struktur, der Inhalt und das Layout der Dokumente kann manipuliert werden.

Dynamic Invocation Interface (DII) - Alternative zur Dienstanforderung über den Stub, der manchmal auch Static Invocation Interface (SII) genannt wird. Hiermit ist es ihm möglich, erst zur Laufzeit des Programms die aufzurufende Operation eines neuen Interfacetyps zu bestimmen und eine Anforderung zu generieren (dynamic typing). Anhand des eingegangenen Requests kann die Objekt-Implementation nicht feststellen, ob der Klient das SII oder das DII benutzt hat.

dynamic typing – Die dynamische Typisierung, also die Bestimmung des Typs einer Variablen erfolgt zur Laufzeit eines Programms. Diese Aufgabe übernimmt das Laufzeitsystem wie z.B. die JVM.

Geokodierung - Einpassen der Bildkoordinaten in ein geodätisches Referenzsystem, d.h. in reale "Weltkoordinaten"

JCP - Der **Java Community Process** ist ein Standardisierungskomitee, das sich speziell der Weiterentwicklung der Programmiersprache Java widmet. Der aktuelle Organisationsablauf wurde selbst durch den JCP definiert in einem gleichartigen Prozess wie die Spracherweiterungen im JSR 215 entwickelt. Jede Erweiterung muss einen bestimmten Prozess durchlaufen. Die Erweiterungen werden Java Specification Request (JSR) genannt und einfach durchnummeriert. Auf der Webseite sind alle JSR's gelistet (URL - http://www.jcp.org/en/jsr/all).

JTWI - JSR 185 - Java Technology for the Wireless Industry: Definiert eine Industriestandardplattform für Java fähige mobile Geräte nächster Generation. Definiert durch Java Community Process(JCP) von einer Gruppe von Experten der leitenden Handy-Hersteller, wireless carriers und Softwarehändler. JTWI spezifiziert die Technologien, die in allen JTWI-fähigen Geräten integriert sein müssen d.h. CLDC 1.0(JSR 30), MIDP 2.0 (JSR 118), und WMA 1.1 (JSR 120), genau wie das CLDC 1.1 (JRS 139) und MMAPI (JSR 135). Die Zersplitterung der APIs wird minimiert.

Oasis (Organization for the Advancement of Structured Information Standards)

Open-Source (Quelloffenheit)- Ein Softwareprodukt wird dann ein open-source Produkt genannt, wenn dessen Quellcode veröffentlicht wird. Meistens ist so ein Produkt dann auch kostenlos.

PDA - Personal Digital Assistant
Es sind kleine Computer, die man dank der kleinen Größe jeder Zeit bei sich tragen kann. Man unterscheidet zwischen so genannten low-end und high-end PDAs.

High-end PDAs sind normalerweise Geräte, die am oberen Ende der Produktlinie platziert sind. Dieses obere Ende zeichnet sich gegenüber dem unteren Ende insbesondere durch den höheren Preis und maximale Funktionalität (mehr Speicherplatz, mehr Prozessorpower, usw. ...) aus. Die maximale Funktionalität impliziert sehr oft "allerneueste und allerbeste Technologie", ist jedoch nicht notwendigerweise daran gebunden.

POI – Point Of Interest
Benutzerdefinierte Objekte, die mit einer Position auf der Erde verknüpft sind. Können im MapPoint über die Adressangabe und 300 zusätzliche Attribute beschrieben und in einer Benutzerdatenbank gespeichert werden.

Primärmetrik punktbezogene Positionsdaten. Die jeweilige Position wird dabei abhängig von ihrer Auflösung streng fixiert.
Anwendungsbereich: Vermessung
Beispiele: ALK (Grenzsteine), Koordinaten bei Vektordaten und Positionsinteger bei Rasterdaten.

SAX (Simple API for XML) – ist ein Standard einer API zum Parsen von XML-Dokumenten. Sie ist im gegensatz zu DOM zustandslos und Ereignis orientiert. Die API wird auf mobilen Geräten eingesetzt. Das zu verarbeitende Dokument muss nicht komplett gespeichert werden, wodurch nicht viel Speicher benötigt wird.

Sekundärmetrik - flächenbezogene Angaben.
Anwendungsbereiche: z.B. amtliche Statistik
Beispiele für Sekundärmetriken: Postleitzahlen, postalische Bezirke, Ortskennziffern des Telefonnetzes, Straßennamen etc.

static typing - Der Interfacetyp muss bereits zur Übersetzungszeit feststehen.

W3C (World Wide Web Consortium) – ist das Gremium zur Standardisierung des World Wide Web betreffender Techniken.

Kapitel 10. Anhang

Entwickeln mit dem MWS:
Zugang beantragen
MWS referenzieren
    Standard WSDL
    http://staging.mappoint.net/standard-30/mappoint.wsdl
    Verschlüsselte WSDL
    https://staging.mappoint.net/secure-30/mappoint.wsdl
SDK Herunterladen

Um die Sicherheit bei der Kommunikation zu erhöhen wird das HTTP Digest Access authentication Protokoll verwendet.

Arten mobiler Anwendungen:
Mobile Office (e-Mail, Kalender, Kontakte, Notizen,...)
Mobile Intranet-Anwendungen (Field Services, Sales Force Automation, Order Services,...)
Mobile Steuerung, Überwachung (Notfall-Services, Wartungs-Services, Überwachungs-Services, Telematik-Services,...)
Mobile Kommunikation (Sprache, SMS, Messaging, Multimedia Services,...)
Mobile Information Services (News, Aktien, Wetter, Verkehr, Push-Services,...)
Mobile Commerce (Banking, Handel, Einkauf, Auktion, Ticketing,...)
Mobile Entertainment (Musik, Video, Kamera, Spiele,...)
location-based services (Reise-Services, Routenführung, Tracking Services, Flotten-Management,...)

Wichtigste Anforderungen bzw. Anfragen an ein GIS:
a. Gib mir die Lage der Entities vom Typ A.
b. Gib mir die Lage von Entity A in Relation zur Umgebung B.
c. Zähle alle Entitäten vom Typ A, deren Entfernung D zu den Entitäten vom Typ B nicht überschritten wird.
d. Berechne die Funktion f() an der Position P.
e. Berechne die Größe von B (Bereich, Umkreis).
f. Ermittle das Ergebnis der Überschneidungen bzw. der Überlappungen verschiedener arten geographischer Daten.
g. Ermittle den Pfad mit den geringsten Kosten, dem geringsten Widerstand oder Entfernung über Grund von X nach Y.
h. Liste von Attributen der Entitäten, die sich an den Punkten P1 und P2 befinden.
i. Ermittle welche Entitäten die nächsten zu denen, mit einer bestimmten Kombination von Attributen, sind.
j. Entitäten mit gewissen Attributen neu ordnen bzw. mit anderen Farben zeichnen.
k. Gib mir den Z-Wert an den Punkten P1..Pn (unter Angabe von einigen bekannten Z-Werten an den Punkten Q1..Qn).
l. Benutze numerische Methoden um neue Attribute bzw. Entitäten aus bereits existierenden abzuleiten.
m. Simuliere die Auswirkung des Prozesses P über die Zeit T für ein vorgegebenes Szenario S, unter Benutzung der digitalen Datenbank als ein Modell der realen Welt.

Erläuterung der Klassen des lcdui-Packages:

Alert

Ist eine Klasse die es ermöglicht, Systemmeldungen dem Benutzer anzuzeigen. Eine Verweildauer lässt sich einstellen. Ein Imageobjekt kann gesetzt werden.

AlertType

Ist eine Hilfsklasse, die den Typ eines Alerts bestimmt. Jeder Typ besitzt andere Eigenschaften, wie z.B. ein akustisches Signal.

Canvas

Ist eine abstrakte Oberklasse, die eine low-level-Grafikbearbeitung ermöglicht. Sie eignet sich besonders für Spieleprogrammierung.

ChoiceGroup

Grupiert auswählbare Elemente einer Form.

Command

Ein Konstrukt, welches die Informationen über eine Aktion kapselt. Es lässt sich an jedes der lcdui-Objekte anhängen.

DateField

Ermöglicht das Anzeigen und Editieren von Datum und Zeit. Es ist eine Unterklasse von Item und kann nur innerhalb einer Form existieren.

Display

Ein Werkzeug, welches das Display verwaltet.

Displayable

Nur ein displayable-Objekt kann auf dem Display angezeigt werden.

Font

Repräsentiert Schriftarten.

Form

Eine Unterklasse von Screen, die unterschiedliche Itemobjekte enthalten kann.

Gauge

Ein Werkzeug zur grafischen Fortschrittsanzeige.

Graphics

Ermöglicht das Rendern zweidimensionaler Grafiken.

Image

Dient zum Laden von Bildern.

ImageItem

Verändert das Layout eines Imageobjektes innerhalb einer Form bzw. eines Alerts.

Item

Eine Superklasse für alle Komponenten, die einer Form hinzugefügt werden können.

List

Ein Screen, das eine Auswahlliste führt.

Screen

Ist eine Superklasse für Objekte, die das Benutzungsinterface zwischen Benutzer und System bilden.

StringItem

Ein Item, welches einen String enthalten kann.

TextBox

Ein TextBoxobjekt ist ein Screenobjekt. Es ermöglicht die Eingabe und das Editieren von Text.

TextField

Ähnlich wie TextBox. Mit dem Unterschied, dass es an einem Formobjekt hängt.

Ticker

Es ist ein Textstreifen, der immer wieder quer über das Display läuft. Es kann an alle Screenobjekte gebunden werden.

Versicherung über Selbständigkeit

Hiermit versichere ich, dass ich die vorliegende Arbeit im Sinne der Prüfungsordnung nach §24(5) ohne fremde Hilfe selbständig verfasst und nur die angegebenen Hilfsmittel benutzt habe.

Hamburg, 30. August 2006 _____

Ort, Datum Unterschrift