D I P L O M A R B E I T

Adressen Web-Service
mit Java Web-Services Developing Framework und SOAP

durchgeführt am
Studiengang für Telekommunikationstechnik und -systeme
der
FH-Salzburg Fachhochschulgesellschaft mbH

vorgelegt von
Bernd Resch

Leiter des Studiengangs:	Prof. Dipl. Ing. Dr. Gerhard Jöchtl
Betreuer:	Dr. Karl Entacher

Salzburg, Februar 2005

Danksagung

An erster Stelle danke ich meiner Familie, die mich in all meinen Lebenswegen, -lagen und Entscheidungen perfekt unterstützt hat.
Weiters möchte ich Manfred Mittlböck einen großen Dank für seine hervorragende und weitsichtige Betreuung meiner Arbeit beim Studio iSPACE - Research Studios Austria, einem Unternehmen der ARC Seibersdorf Research GmbH aussprechen. Außerdem möchte ich Karl Entacher für seine hilfreiche Unterstützung als Betreuer an der FH danken.
Abschließend danke ich noch all meinen Freunden, die mir während der Erstellung dieser Diplomarbeit als Anlaufpunkte bei Problemen jeglicher Art zur Seite gestanden sind.

Eidesstattliche Erklärung

Hiermit versichere ich, Resch Bernd, geboren am 27. März 1981 in St. Michael/Lg., dass die vorliegende Diplomarbeit von mir selbstständig verfasst wurde. Zur Erstellung wurden von mir keine anderen als die angegebenen Quellen und Hilfsmittel verwendet.

Salzburg, am 18.02.2005

_____ _____

Resch Bernd Matrikelnummer

Informationen

Vor- und Zuname:	Bernd Resch
Institution:	FH-Salzburg Fachhochschulgesellschaft mbH
Studiengang:	Telekommunikationstechnik und –systeme
Titel der Diplomarbeit:	Adressen Web-Service mit Java Web Services Developing Framework und SOAP
Betreuer an der FH:	Dr. Karl Entacher
Betreuer in der Firma:	Mag. Manfred Mittlböck, iSPACE

Schlagwörter

1. Schlagwort:	Adressen Web-Service
2. Schlagwort:	SOAP
3. Schlagwort:	Java Web Service Developer Pack

Abstract

This project documents the result of the project *Adressen Web-Service mit Java Web Services Developing Framework und SOAP*, which originated as a result of the demand for a standardised exchange of addresses and coordinates, which is currently not realised in a Europe-wide extent. The main condition for the proposed concept was its implementation on the basis of SOAP, which should be used as the transport protocol for the application.

The thesis first presents a web service, developed in the course of the project, which maps geographical positions defined by an x- and a y-coordinate to a postal address by performing a dynamic database search. Additionally to this principal application, several supplementary features have been implemented in order to bring the service one step closer to practical usability. Such extensions are e.g. a circumcircle search around a point, a Universal Unique Identifier (UUID) for user authentication, different tolerance mechanisms concerning the address input, a configuration file with pre-defined parameters for the database connection, values for the compass search etc. and numerous others. Furthermore, the whole web service is packed in a web application archive, which contains all files such as JSP files, servlets, HTML and XML files a.s.o. Thus, the service can be easily installed on any web server.

The thesis concludes by giving an outlook to potential further developments to the web service by using the Web Service Description Language (WSDL) and the Universal Description, Discovery and Integration (UDDI) as well as by an integration of a coordinate transformation system.

Inhaltsverzeichnis

Abbildungsverzeichnis

Tabellenverzeichnis

1 Einleitung

Das Projekt „Adressen Web-Service mit Java Web Service Developing Framework (SOAP)“, in der Folge kurz mit „*adr_soap*“ bezeichnet, gründet auf der Idee, eine standardisierte Variante eines Web-Service-basierten Adress-Mapping Systems zu erstellen.
Ziel dieses Projektes ist eine Adress- bzw. Koordinatenabfrage, die unabhängig vom Typ des Abfragegerätes funktionieren soll. Diese sehr allgemein gehaltene Implementierung der Kommunikation und vor allem die konsistente Definition der SOAP-Schnittstelle sollen darüber hinaus als Basis für eine mögliche Standardisierung des Austausches geografischer Adressinformation auf Basis der in Österreich diskutierten Adressschreibweise dienen.
Die Modularität des Web Service besteht darin, dass die Schnittstelle des Frontends zur eigentlichen Datenbankabfrage am Web-Server so konsistent definiert ist, dass fertig verpackte Nachrichten geschickt werden können. Dadurch, dass viele Embedded Devices, wie sie vielfach in satellitengestützten Notfallfahrzeugen verwendet werden, ihre Anfragen über das SOAP übertragen, liegt ein Fokus bei der Entwicklung einer Datenschnittstelle nach außen auf Grundlage dieses standardisierten Übertragungsprotokolls.

Im Zuge dieser Diplomarbeit werden die für die Implementierung verwendeten Technologien erläutert und die technische Umsetzung dargestellt. Darin enthalten sind Beschreibungen der Grundstruktur sowie zusätzlich implementierter Features. Danach folgen Erläuterungen zu den einzelnen Modulen, die das Web-Service umfasst, also zu Konfigurationsfile, JSP Seiten und Servlets. Darauf folgend werden einige spezielle Details zur Umsetzung diskutiert, was der besseren Verständlichkeit des Implementierungskonzeptes dienen soll. Kapitel 7 umfasst dann Informationen, die notwendig sind, um das Web-Service nutzen zu können.
Um kundenspezifische Anforderungen bereits bei der Konzeption miteinzubeziehen, wurde bei der Entwicklung besonderes Augenmerk auf eine möglichst rasche Fertigstellung eines diskutierbaren Prototyps gelegt. Dies ermöglicht die Integration verschiedenster individueller Konfigurationen.
Abschließend wird ein Überblick über mögliche Weiterentwicklungen der Anwendung gegeben.

2 Verwendete Technologien

Für das *adr_soap*-Projekt werden mehrere standardisierte Technologien und dazugehörige Programme verwendet, die vor der Erläuterung der technischen Implementierung beschrieben werden.

2.1 Extensible Markup Language

Die Extensible Markup Language (XML) ist eine Meta-Auszeichnungssprache, die ähnlich der Hypertext Markup Language (HTML) in einer Tag-Struktur aufgebaut ist. Im Gegensatz zu HTML können in XML-Tags jedoch zusätzlich individuell definiert werden, was in größerer Flexibilität, Vielfältigkeit und damit höherer Effizient in der Programmierung resultiert.
XML beschreibt die Struktur und den Inhalt eines Dokuments, nicht aber die Formatierung, welche durch einen Style Sheet, in den Unterkapiteln 2.1.4 und 2.1.6 beschrieben, bestimmt wird. Das Dokument beinhaltet also Element-Tags und keine formale Beschreibung.

2.1.1 Vorteile aus Entwicklersicht

Für Applikationsentwickler bietet XML eine Reihe von Vorteilen, die im Folgenden dargestellt werden. Erstens können auf XML basierend *domänenspezifische* Auszeichnungssprachen in verschiedensten Bereichen wie Mathematik, Musik oder auch Geografie definiert werden. Damit entsteht die Möglichkeit, effiziente Dokumente mit wenig Redundanz zu schaffen, die auch für nicht-fachkundige Personen verständlich sind. Weiters sind gezielt erstellte XML-Dokumente weitgehend *selbsterklärend*, was wiederum Lesbarkeit für breite Zielgruppen und über lange Zeit sichert. Außerdem ist XML als frei verfügbare und nicht urheberrechtlich geschützte Sprache prädestiniert für einen *standardisierten Daten-austausch* zwischen diversen Applikationen. Diese Eigenschaft garantiert auch eine plattform- und applikationsunabhängige Verwendung von XML, da keine verschiedenen Formate wie z.B. bei Textverarbeitungsprogrammen (.doc, .tex, .rtf) verwendet werden, die meist undokumentiert sind und sich laufend ändern. Schließlich eignet sich XML auf Grund seines *strukturierten Aufbaus* hervorragend für die Verarbeitung von großen Datenmengen, nachdem nicht nur Elemente, sondern auch ihre Beziehungen zueinander definiert werden können. Überdies enthält XML Mechanismen, mehrere externe Quellen einzubeziehen und die Resultate in einem Dokument auszugeben.

2.1.2 Zugrunde Liegende Technologien

XML funktioniert auf Grundlage von diversen Technologien wie HTML, Cascading Style Sheets (CSSs), der Extensible Style Language (XSL), Uniform Resource Locators (URLs) und Uniform Resource Identifiers (URIs) sowie der Extensible Linking Language (XLL) und dem so genannten Unicode Zeichenset.
Auf Grund seines Status als meistverwendete Auszeichnungssprache wird HTML an dieser Stelle nicht mehr gesondert behandelt. Mehr Information über die Hypertext Markup Language kann in [MINT03] gefunden werden. XSL wird in Abschnitt 2.1.6 näher beschrieben.

Als URLs werden standardmäßige Internetadressen wie z.B. *http://www.researchstudio.at* bezeichnet. Trotz der breiten Unterstützung von URLs verwendet XML die generelleren URIs, die sich mehr auf die Ressource selbst und weniger auf ihren Standort beziehen. Generell betrachtet, können URIs die näheste Kopie eines Dokumentes finden.
Das Unicode Zeichenset ist eine Menge von Symbolen, mit Hilfe derer XML-Daten dargestellt werden können, wozu jedoch drei Dinge notwendig sind:

- Ein Zeichensatz für die das Dokument
- Eine Schriftart für die Symbolmenge
- Ein Betriebssystem, das den Zeichensatz unterstützt

CSS und XSL sind in den Abschnitten 2.1.4 und 2.1.6 näher beschrieben.

2.1.3 Elemente und Attribute

Wie bereits erwähnt verwendet XML wie HTML eine Tag-Struktur. Die End- und Verzweigungspunkte der Baumstruktur, die ein XML-Dokument darstellt, werden Elemente genannt. Diese Elemente können optional mit Attributen versehen werden, um es zusätzlich zu beschreiben. Eine beispielhafte Schachtelung von Elementen unter Verwendung von Attributen ist in den folgenden Zeilen dargestellt.

```
<Buch Genre="Fachbuch">
   <Autor Nationalitaet="Deutschland">
      <Vorname>Stefan</Vorname>
      <Nachname>Mintert</Nachname>
   </Autor>
   <Titel>XHTML, CSS und Co.</Titel>
   <ISBN>3827318726</ISBN>
</Buch>
```

Prinzipiell gibt es keine festen Regeln, wie und ob Elemente oder Attribute verwendet werden müssen. Je nach Art und Aufbau der Applikation haben sowohl Kindelemente als auch beigefügte Werte Vorteile. Generell kann jedoch festgehalten werden, dass Daten in Elementen gespeichert werden sollten, Meta-Daten, also Informationen über die Daten selbst, jedoch in Attributen. Die Unterscheidung zwischen Daten und Meta-Daten kann dadurch erfolgen, ob der Leser einen bestimmten Informationsteil erwartet bzw. ob die Elementstruktur noch gewährleistet ist. Überdies muss in Betracht gezogen werden, dass Elemente in Hinblick auf zukünftige Weiterentwicklungen leichter erweiterbar sind.
Attribute können verwendet werden, wenn Werte eindimensionale Einheiten darstellen, die nicht mehr in Unterelemente unterteilt werden können, wie beispielsweise die Höhe und die Breite eines Bildes. Eine weitere Einsatzmöglichkeit von Attributen besteht bei der Speicherung von Informationen, die nicht unmittelbar mit dem Inhalt des Dokuments zu tun haben, wie z.B. IDs. Außerdem werden oft auch dokumentspezifische Formatierungsinformationen in Attributen gespeichert, was eine Dokumentmodifikation ohne Veränderung des zu Grunde liegenden Style Sheets ermöglicht.
Abschließend soll nochmals erwähnt werden, dass erst durch Prüfung der Applikationsanforderungen eine Unterscheidung zwischen Daten und Meta-Daten getroffen werden kann.

2.1.4 Prozesse in XML

Der *Parser*, auch *Prozessor* genannt, liest eine XML-Struktur und prüft sie auf Wohlgeformtheit – wie in Abschnitt 2.1.9 beschrieben, wobei optional eine Kontrolle auf Gültigkeit erfolgen kann. Wenn die Struktur diese Tests passiert, wandelt der Prozessor das Dokument in einen Elementbaum um, welcher dann an eine Endapplikation wie beispielsweise einen Webbrowser oder ein anderes Programm, das XML-Daten verarbeiten kann, übergeben wird. Der Lebenszyklus eines XML-Dokumentes ist in Abbildung 2.1 dargestellt.

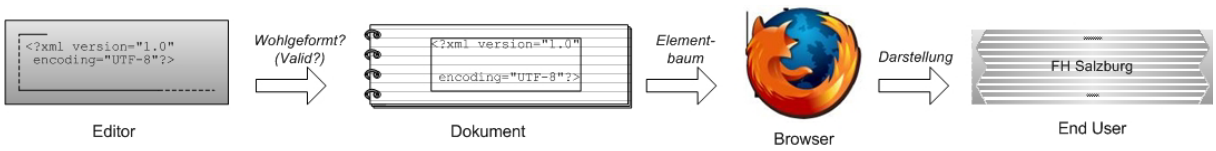

Abbildung 2.1: Lebenszyklus eines XML-Dokumentes, nach [HARO99].

2.1.5 Cascading Style Sheets

Cascading Style Sheets (CSSs) dienen dazu, dem Browser mitzuteilen, wie XML-Daten darzustellen sind. CSSs müssen also mit einem Dokument mit gesendet werden, damit dieses

fehlerfrei angezeigt werden kann. CSSs, die ursprünglich für HTML entworfen wurden, legen Formateigenschaften wie Schriftgröße und –familie, Absatzeinzug und –ausrichtung und andere Parameter fest, wobei mehrere Style Sheets auf ein Dokument angewendet werden können.

2.1.6 Extensible Style Language

XSL dient wie CSS dazu, XML-Daten formatiert darzustellen, wobei XSL im Gegensatz zu CSS auch Attribute anzeigen kann. XSL beinhaltet eine Menge von Regeln, die bestimmten Mustern in einem XML-Dokument zugeordnet sind. XSL-Dateien sind ihrerseits wiederum wohlgeformte XML-Dokumente. Ein XSL-Prozessor liest XML-Daten und versucht, Muster, die im Style Sheet festgelegt sind, wieder zu finden. Wenn solch ein Muster erkannt wird, werden analog der XSL-Regel bestimmte Textkombinationen ausgegeben.
Prinzipiell besteht XSL aus zwei Teilen; der erste ermöglicht mittels bestimmter Regeln eine Transformation von Tags in andere Tags. Damit können XML- in HTML-Dokumente umgewandelt werden, indem standardisierte HTML-Tags, gegebenenfalls mit Attributen, verwendet werden. Zweitens ermöglicht XSL eine formatierte Ausgabe von Dokumenten als Seiten z.B. in Browsern. Die XSL-Formatierung ermöglicht also, die Erscheinung und das Layout einer Seite festzulegen.
Mittels XSL-Transformationen (XSLT) können XML-Dokumente in diverse Formate umgewandelt werden, um auf Endgeräten wie Handies oder Personal Digital Assistants (PDAs) fehlerfrei angezeigt werden zu können. XSLT ähnelt eher einer Programmiersprache als einer bloßen Vorgabe der Dokumentstruktur, nachdem über die reinen Darstellung von Inhalten hinaus auch Abfragen inkludiert und damit variable Ausgaben erzeugt werden können.

2.1.7 XPath, XPointer, XLinks

Drei weitere Technologien, die an dieser Stelle der Vollständigkeit halber erwähnt, jedoch nicht im Detail erklärt werden sind XPath, XPointer und XLinks.
Mittels XPath können ausgewählte Informationsblöcke innerhalb eines XML-Douments, wobei zwischen sieben verschiedenen Elementen unterschieden wird: Wurzelelement, Element, Text, Attribut, Kommentar, Verarbeitungsanweisungen und Namensräumen. Im Gegensatz zu XSL, mit der „Schablonen“ über Dokumente gelegt werden können, kann XPath modular bestimmte Elementblöcke ansprechen.
Die Extensible Style Language (XLL) besteht aus zwei Teilen, XLink und XPointer. Mit Hilfe von XLink kann definiert werden, wie ein XML-Dokument zu einem anderen verlinkt

ist. XLink verwendet jedoch URIs als Ziele, was zur Einschränkung führt, dass nur das gesamte Dokument Ziel des Links sein können.
XPointer baut – so wie XSL und das XML Schema - auf XPath auf und erlaubt die Adressierung von bestimmten Punkten bzw. Fragmenten innerhalb eines Dokumentes. Die Identifikation dieser Fragmente erfolgt wie in XPath mittels eines Vergleiches von Zeichenketten. Die Verwendung von XPointer ermöglicht eine Verlinkung zu Teilen eines anderen Dokumentes, ohne dieses z.B. durch manuelles Einfügen von Ankern verändern zu müssen.

2.1.8 CSS oder XSL

Im Generellen erfüllen CSS und XSL die gleichen Aufgaben, wobei zweitere Technologie auf Grund der möglichen Verwendung von Abfragen, Schleifen, etc. mit Sicherheit die mächtigere ist. Die Ausgabe des Dokumentinhaltes geht in XSL – nicht wie bei der Verwendung von CSS – über die bloße Darstellung von Text inklusive dessen Formatierungsregeln hinaus. CSS kann nur auf einer elementweisen Basis angewendet werden während XSL Elemente jedoch neu strukturieren kann, wobei CSS den Vorteil hat, von einer breiteren Palette von Browsern unterstützt zu werden, XSL jedoch flexibler ist und XML-Dokumente relativ einfach in HTML-Strukturen konvertiert werden können. Zusammenfassend kann also festgestellt werden, dass auf lange Sicht XSL verwendet werden sollte, um eine flexible und konsistente Darstellung von XML-Daten zu gewährleisten.

2.1.9 Wohlgeformtheit eines XML-Dokumentes

Wohlgeformtheit ist die Minimalanforderung an ein XML-Dokument, um weiter verarbeitet und dargestellt werden zu können und bedeutet, dass das Dokument folgenden Regeln gerecht wird:

- Am Anfang des Dokumentes hat die XML-Deklaration zu stehen (minimal: `<?xml version="1.0"?>`)
- Das Dokument muss mindestens ein Datenelement enthalten
- Es muss ein äußerstes Element geben, das alle weiteren beinhaltet [MUEN01]

Diese Wohlgeformtheit besitzt große Vorteile gegenüber teilweise unsauber programmierten HTML-Seiten. Erstens ist ein wohlgeformtes Dokument leichter zu lesen und damit einfacher zu erweitern und zweitens erreichen automatisierte Suchmaschinen eine höhere Effizienz.

2.1.10 Document Type Definitions

Eine Document Type Definition (DTD) wird benötigt, wenn ein XML-Dokument nicht nur wohlgeformt, sondern auch gültig, also „validierbar“ gegen eine DTD, sein soll. Eine DTD legt eine Liste von Elementen, Attributen, Entitäten[1] und deren Beziehung zueinander fest.
Es bestehen zwei Möglichkeiten, DTDs zu definieren. Erstens können sie direkt ins XML-Dokument eingebunden werden oder sie können in einer eigenen Datei erstellt und dann mittels eines Verweises in das XML-File eingebunden werden. Der Vorteil von externen DTDs besteht darin, dass sie von mehreren XML-Dokumenten gleichzeitig verwendet werden können.
DTDs legen also fest, welche Elemente und Strukturen in einem XML-Dokument erlaubt sind. Außerdem kann durch die Verwendung von DTDs der Aufbau eines Dokumentes veranschaulicht werden ohne die Daten selbst preisgeben zu müssen.
Das folgende Beispiel dient der Veranschaulichung der Verwendung einer DTD:

Ohne DTD	Mit DTD

`<?xml version="1.0"?>`
`<university>`
`  FH Salzburg`
`</university>`
	`<?xml version="1.0"?>`
	`<!DOCTYPE universtiy [`
	`  <!ELEMENT university (#PCDATA)>`
	`]>`
	`<university>`
	`  FH Salzburg`
	`</university>`

Tabelle 2.1: XML-Beispiel ohne und mit DTD.

Wie aus Tabelle 2.1 ersichtlich unterscheiden sich die beiden Dokumente nur durch die Codezeilen

```
<!DOCTYPE universtiy [
 <!ELEMENT university (#PCDATA)>
]>
```

Diese Dokumenttypdeklaration wird zwischen der XML-Deklaration und dem ersten Element notiert und bildet zusammen mit der XML-Deklaration den so genannten Prolog.

In der gleichen Manier wie DTDs werden auch CSSs in den Quelltext eingebunden:

```
<?xml version="1.0"?>
<?xml-stylesheet type=”text/css” href=”university.css”?>
```

[1] Entitäten sind physische Speichereinheiten, aus denen sich ein XML-Dokument zusammensetzt.

```
<!DOCTYPE universtiy [
 <!ELEMENT university (#PCDATA)>
]>
<university>
 FH Salzburg
</university>
```

Harold [HARO99] bseschreibt detailliert, wie CSSs in XML verwendet und eingebaut werden können.

Um ein valides XML-Dokument zu kreieren, müssen alle verwendeten Elemente in der dazugehörigen DTD deklariert werden. Prinzipiell gilt die Regel: alles, was nicht explizit erlaubt ist, ist verboten. So muss für jedes Element angegeben werden, ob es beliebig oft, mindestens ein Mal oder niemals im Dokument vorkommen darf, was mittels der Zeichen `*`, `+` und `?` geschieht. Attribute werden in einer DTD mittels des `<!ATTLIST>`-Tags deklariert. Mehr Informationen darüber wird in [HARO99] und [MUEN01] zur Verfügung gestellt.

Um eine Auswahl von vorgegebenen Kindelementen anzuführen, wird das Symbol `|` verwendet. Die Ausbildungsstätte also auf die Werte `Universität` und `FH` zu begrenzen, sähe folgendermaßen aus:

```
<!ELEMENT School (University | FH)>
```

Dies bedeutet, dass das Element `School` ausschließlich die Kindelemente `University` oder `FH` beinhalten kann.

Mehr über die praktische Anwendung von DTDs sowie eine generelle praktischere Annäherung an XML können in [HARO05], [STLA98] und [MUEN01] gefunden werden.

2.1.11 Schema Definition

Die XML Schema Definition (XSD) bietet als Ersatzentwicklung für DTD einen mächtigeren Mechanismus, XML-Dokumente zu validieren. XSD ermöglicht sowohl, die Syntax eines Dokumentes festzulegen, als auch eigene Datentypen zu definieren, was bei der Dokument-validierung auch die Erkennung von semantischen Fehlern möglich macht. Außerdem erlaubt XSD die Verwendung von mehreren Namensräumen in einem Schema und unterstützt Vererbung, listenbasierte Datentypen u.v.m. Mehr über die praktische Anwendung von XSD kann unter [HEIN03] gefunden werden.

2.1.12 XMLSpy

Für die XML-Programmierung wird mit *XMLSpy* ein Entwicklungstool mit einer Fülle von Möglichkeiten verwendet. Aufgrund der vorhandenen Lizenz von iSPACE wurde dieses mit einer grafischen Entwickleroberfläche versehene Programm verwendet. Informationen über die volle Auswahl der Features werden auf [ALTO04] bereitgestellt.

2.2 Geography Markup Language

Die Geography Markup Language (GML) ist die Realisierung eines XML-Schemas, das zur Modellierung, zum Austausch und zur Speicherung von geografischen Daten dient. GML stellt unzählige Objekte zur Beschreibung von Koordinatenreferenzsystemen, geografischen Elementen (Punkten, Linien, …), Topologien, Maßeinheiten uvm. zur Verfügung.
Ursprünglich ist GML unter Berücksichtigung folgender Designprinzipien entworfen worden:

- Schaffung eines herstellerunabhängigen Rahmens für die Darstellung und den Austausch von geografischen Objekten
- Grundlage für Speicherung und Portierung von Applikationsschemata und Datensätzen
- Unterstützung der Beschreibung domänenspezifischer Schemata

2.2.1 GML 3: Eigenschaften

Die aktuelle dritte Version von GML entstand auf Grund von mehreren Mankos von GML 2 wie beispielsweise die Restriktion auf einfache geometrische Strukturen, die durch zwei Koordinaten definiert sind. Um komplexere Strukturen darzustellen, musste in dieser Version eine Ansammlung von Features[2] ein Objekt beschreiben.
Mit GML Version 3 wird nun versucht, ihren Vorgänger in folgenden Punkten zu verbessern:

- Darstellungsmöglichkeit von komplexen Strukturen wie nicht-linearen 3D-Objekten oder dynamischen Features mit zeitlichen Eigenschaften
- Unterstützung von räumlichen und zeitlichen Referenzsystemen und Einheiten
- Standardisierte Visualisierung von Features

Um diesen Anforderungen gerecht zu werden, musste die Palette an Basisschemata auf die achtfache Größe der Vorgängerversion ausgedehnt werden. Trotz aller Erweiterungen wurde

[2] In GML bezeichnet ein Feature eine Abstraktion von Vorgängen in der „echten“ Welt

eine Abwärtskompatibilität der jetzigen Version 3.1.0 mit allen Vorgängern bis GML 2.1.2 aufrecht erhalten [COX04].

2.2.2 Klassenhierarchie

Abbildung 2.2 illustriert die Klassenhierarchie in GML 3.1.0 mittels einer Unified Markup Language (UML) Struktur.

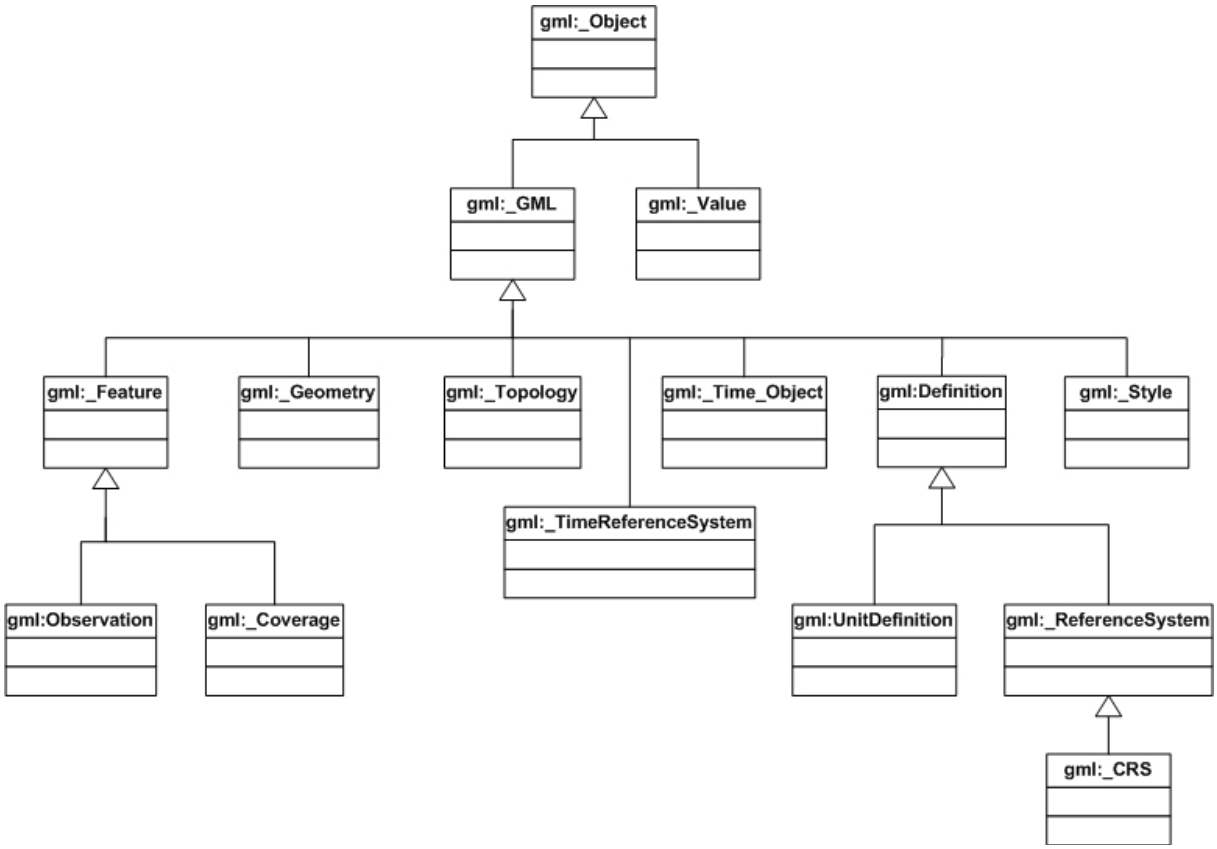

Abbildung 2.2: Klassenhierarchie in GML 3.1.0, nach [COX04].

Die Elemente mit voranstehendem Unterstrich können als repräsentative Objekte ihrer Klasse angesehen werden. So steht z.B. *gml:_Topology* für jegliche Art einer GML-Topologie. Die einzelnen normativen GML-Schemata befinden sich rund um diese Objekte.

2.2.3 Schemata

Die GML-Struktur baut auf zahlreichen Schemata auf, die es ermöglichen, verschiedene Arten von geografischen Daten darzustellen.

XLinks, ein Teil von XLL und in Abschnitt 2.1.7 kurz erläutert, definiert wie zwei XML-Dokumente miteinander verknüpft sind, indem es auf einen bestimmten URI zeigt, der den Speicherort einer Ressource angibt. In GML können Elemente vier verschiedene XLink-Attribute beinhalten:

- *href*: Identifikator der Ressource, die das Ziel der Assoziation ist
- *role*: Rolle bzw. Beschaffenheit der Zielressource
- *arcrole*: Beschreibung der Rolle der Zielressource im Verhältnis zur aktuellen Ressource
- *title*: Beschreibung der Assoziation oder der Zielressource

Die ersten drei XLink-Attribute werden mittels eines URI dargestellt, das vierte als Text. Die gesamte XLink-Spezifikation kann unter [HARO99] nachgelesen werden.

Das World Wide Web Consortium (W3C) XML-Schema Teil 2 [BIRO04] definiert zahlreiche „einfache“ Typen, mit Hilfe derer Werte ohne eigene Auszeichnung als Buchstabensymbole dargestellt werden können. Nachdem GML durch XML-Schemata definiert wird, werden diese einfachen Typen so weit wie möglich verwendet. Falls das W3C XML-Schema keine passenden Typen beinhaltet, können so genannte GML-Basistypen daraus abgeleitet werden. Diese sind in Annex C des Dokumentes [COX04] detailliert beschrieben.

2.2.3.1 Das gmlBase Schema

Das *gmlBase* Schema definiert Komponenten, die das GML-Modell formen:

- Ein Basis XML-Typ, von dem alle GML-Objekte abgeleitet werden können
- Ein Muster und Komponenten für GML-Eigenschaften
- Muster und Komponenten für Feldarrays
- Komponenten für die Verbindung von Metadaten mit GML-Objekten
- Komponenten für die Erstellung von Definitionen und Nachschlagewerken

Das Basiselement, dem alle anderen Elemente untergeordnet sind, trägt den Namen *gml:_Object*, wie aus Abbildung 2.2 ersichtlich ist. Dieses Element hat keinen definierten Typ und ist deshalb automatisch ein XML-Schema namens „anyType“. Um ein identifizierbares GML-Objekt zu schaffen, steht das abstrakte Element *gml:_GML* zur Verfügung, das alle anderen GML-Elemente beinhaltet. Dieses Element gepaart mit der Eigenschaft „gml:AbstractGMLType“ bildet ein in GML-Schemata oft verwendetes Muster, wobei jedes einzelne GML-Objekt durch eine globale Elementdeklaration dargestellt wird, die eine Assoziation zu einem XML-Schema hat. Die Unterelemente, die in Abbildung 2.2 aufgelistet sind, werden in [COX04] näher beschrieben.

2.2.3.2 GML-Eigenschaften

Ein GML-Element kann nur dann eine GML-Eigenschaft sein, wenn es ein Kindelement des Wurzelelementes *gml:_Object* ist. Diese Eigenschaften können durch zwei Methoden beschrieben werden. Erstens kann die Eigenschaft durch den Inhalt des Elements direkt beschrieben werden oder sie ist an einem anderen Ort verfügbar, der über ein *xlink:href*-Attribut im Eigenschaftselement definiert wird.
Neben dem Basisschema bestehen in GML noch weitere Schemata, die in der folgenden Auflistung beschrieben sind:

- *Modell von Merkmalen*: aussagekräftige Objekte gegliedert in zahlreiche Bereiche
- *Grundlegende Geometrie*: Basiselemente, Aggregate, komplexe Elemente
- *Andere geometrische Elemente*: zusätzliche geometrische Elemente, um Objekte aus der realen Welt darzustellen, die ein komplexeres geografisches Modell bedingen
- *Geometrische komplexe Zusammensetzungen*: Aggregate von geometrischen Elementen; diese Aggregate besitzen kein zusätzlichen internen Strukturen
- *Koordinatenbezugssysteme*: beinhaltet diverse geodetische und nicht-geodetische Coordinate Reference Systems (CRSs)
- *Topologien*: Beschreibung von Objekteigenschaften, wobei die Objekte unter kontinuierlicher Veränderung unveränderlich sind; z.B. entsteht eine Ellipse durch Streckung eines Kreises → Beschreibung des Verhältnisses zwischen Objekten
- *Zeitinformationen und dynamische Features*: Beschreibung der Reihenfolge oder der Eigenschaften von Features; zeitabhängige Eigenschaften von Elementen
- *Definitionen und Stichworte*: Begriffsdefinitionen und eine Sammlung dieser
- *Maßeinheiten*: Beschreibung von quantitativen Werten mit Referenzskalen
- *Richtungen*: Richtungs- und Orientierungsbeschreibung
- *Beobachtungen*: Modellierung von Beobachtungen und Beschreibung von Metadaten über einen Datenerfassungsvorgang, z.B. eine Aufnahme mit einer Kamera
- *Oberflächen*: Abbildung eines Raum-Zeit-Gefüges auf eine diskrete Bereichsmenge
- *Formgestaltungen*: dient zur Trennung von Datensätzen und deren Darstellungs-möglichkeiten, Gestaltungen können also den Daten selbst hinzugefügt werden

Die Elemente der oben genannten Schemata sind in Annex C in [COX04] aufgelistet und detailliert beschrieben. In diesem Projekt wird das *gml:Point*-Schema, das Teil der grundlegenden Geometrie ist, beispielsweise folgendermaßen verwendet:

```
<gml:Point srsName="EPSG:4326">
```

```
      <gml:pos>16.0391074540754 47.775758835661  </gml:pos>
   </gml:Point>
```

Das *gml:Point*-Element enthält ein Attribut namens *srsName*, das den European Petroleum Survey Group (EPSG) Code des verwendeten Koordinatensystems beinhaltet. Weiters umfasst das Element ein Kindelement, nämlich *gml:pos*, das Werte für x-, y- und z-Koordinaten, jeweils getrennt durch ein Leerzeichen, enthält, wobei in diesem Fall keine z-Koordinate vorhanden ist.

In diesem Projekt wird GML zur standardisierten Darstellung von geografischen Punkten verwendet. Beispiele für die Kommunikation über SOAP werden in Kapitel 5 gegeben. Mehr über die Hintergründe, Definitionen, Schemas und deren Verwendung kann in [COX04] gefunden werden.

2.3 SOAP

SOAP, früher ein Akronym für Simple Object Access Protocol, ist ein plattformunabhängiges Übertragungsprotokoll für den Informationsaustausch von strukturierten Daten mit relativ geringem Overhead. Das auf XML basierende Protokoll besteht aus drei Teilen: die *Envelope* definiert den generellen Hintergrund einer Nachricht, also was sie beinhaltet, an wen sie adressiert ist, etc. Die so genannten *Encoding Rules* umfassen einen strukturell angeordneten Datenaustausch von anwendungsspezifischen Datentypen. Die *RPC representation* schließlich stellt eine Konvention für Remote Procedure Calls (RPCs) und Antwortnachrichten dar.

2.3.1 Aufbau einer Nachricht

Eine SOAP-Nachricht besteht aus drei Komponenten, aus *Envelope*, *Header* und *Body* [HEIN03], [GUDG03a]. Ein Nachrichtenbeispiel ist in den folgenden Zeilen dargestellt.

```
<SOAP-ENV:Envelope xmlns:SOAP-
ENV="http://schemas.xmlsoap.org/soap/envelope/">
   <SOAP-ENV:Header compass="false" control="1" sid="429714959"/>
   <SOAP-ENV:Body>
      <coords:getAddressResponse
      xmlns:coords="://mssql.researchstudio.at:8081">
         <rowCount>15</rowCount>
         <buffer_val>50</buffer_val>
         <gml:location xmlns:gml="http://www.opengis.net/gml">
            <gml:Point srsName="EPSG:31289">
               <gml:pos>-22049 292997  </gml:pos>
            </gml:Point>
         </gml:location>
```

```
            <STR_NAME>Ainringweg</STR_NAME>
            <HNR>13</HNR>
            <HNR_ALPHA>B</HNR_ALPHA>
            <empty> </empty>
            <ZIP>5020</ZIP>
            <CITY>Salzburg</CITY>
            <empty> </empty>
            <BEZIRK>Salzburg</BEZIRK>
            <empty> </empty>
            <COUNTRY>Österreich</COUNTRY>
            <empty> </empty>
            <empty> </empty>
            <gml:location xmlns:gml="http://www.opengis.net/gml">
               <gml:Point srsName="EPSG:4326">
                  <gml:pos>16.0391074540754 47.775758835661  </gml:pos>
               </gml:Point>
            </gml:location>
         </coords:getAddressResponse>
      </SOAP-ENV:Body>
   </SOAP-ENV:Envelope>
```

2.3.1.1 Envelope

Die SOAP-Envelope ist das Wurzelelement der Nachricht und definiert die XML-Struktur als SOAP-Nachricht, die folgende Bestandteile enthält:

- Einen Namen der Envelope selbst
- Einen Namensraum, der z.B. „*http://www.w3.org/2003/05/soap-envelope*“ lauten kann
- Optionale Attribute, die durch Namensräume qualifiziert werden
- Ein oder zwei Kindelemente (optionaler Header und Body)

Weiters kann die Envelope noch das *encodingStyle*-Attribut beinhalten, das Regeln festlegt, wie die Bestandteile der SOAP-Nachricht angeordnet werden.

Ein selbsterklärendes Beispiel, wie die Envelope in diesem Projekt eingesetzt wird, ist im Folgenden dargestellt:

```
<SOAP-ENV:Envelope xmlns:SOAP-
 ENV="http://schemas.xmlsoap.org/soap/envelope/">
</SOAP-ENV:Envelope>
```

2.3.1.2 Header

Wenn ein Header verwendet wird, muss er das erste Kindelement der Envelope sein. Er kann aus folgenden Teilen bestehen:

- Einem Namen des Headers

- Einem Namensraum, nämlich „*http://www.w3.org/2003/05/soap-envelope*“ wie oben
- Optionale Attribute und Kindelemente

Jedes Kindelement des Headers wird als Header-Block bezeichnet, dessen Eigenschaften in [GUDG03a] beschrieben sind. Weiters ist in diesem Teil der SOAP-Nachricht die Rolle des Knotens definiert. Die Rollen, in denen ein Knoten operieren kann sowie das *mustUnderstand*-Attribut, sind in Unterkapitel 2.3.2 beschrieben.

Die folgenden Codezeile illustrieren die Verwendung eines Headers in diesem Projekt:

```
<SOAP-ENV:Header compass="false" control="1" sid="429714959"/>
```

Der Header ist in diesem Fall ein leeres Element (`<elementName/>`) und beinhaltet drei Attribute, nämlich Werte für die Umkreissuche, die Authorisierung und eine Session-ID.

2.3.1.3 Body

Der Körper der SOAP-Nachricht, der beliebig viele Attribute und Elemente umfassen kann, enthält die eigentliche Nachricht, die an den Empfänger übermittelt werden soll. Der Name des Körpers sollte über einen Namensraum qualifiziert sein, um möglichst eindeutige und aussagekräftige Fehlermeldungen zu erhalten. Wie das Body-Element praktisch verwendet werden kann, ist in den folgenden Zeilen, die auch in der Implementierung dieses Projektes vorkommen, veranschaulicht:

```
<SOAP-ENV:Body>
   <coords:getAddressResponse
   xmlns:coords="://mssql.researchstudio.at:8081">
      <...>
   </coords:getAddressResponse>
</SOAP-ENV:Body>
```

Der Body muss also alle Elemente umfassen, die relevante Informationen beinhalten. Nachdem das Body-Element geschlossen wurde, wird nur mehr das Envelope-Element geschlossen, wie oben gezeigt.
Ein Kindelement des Körpers wird direkt in SOAP definiert, nämlich der *SOAP Fault*, das mehrere mögliche Fehler in Zusammenhang mit der Codierung, der Syntax oder der Rolle der Nachricht bzw. des Knotens abfangen kann. Mehr über diesen Mechanismus kann in [REFS05b] und in [GUDG03a] gefunden werden.

2.3.2 Prozessmodell

Während der Kommunikation über SOAP haben alle beteiligten Knoten eine bestimmte Rolle, die sich über den gesamten Ablauf des Prozesses nicht ändern darf. Prinzipiell sind in SOAP drei Rollen definiert:

- *next*: Jeder Knoten, der am Kommunikationsprozess teilnimmt, muss in dieser Rolle agieren.
- *none*: Kein Knoten, der am Kommunikationsprozess teilnimmt, darf in dieser Rolle operieren.
- *ultimateReceiver*: Der Endknoten, also der schlussendliche Empfänger der Nachricht, muss in dieser Rolle agieren.

Die Rolle wird im Header festgelegt, wie im folgenden Beispiel illustriert, das dem Knoten die Rolle *next* gibt:

```
<SOAP-ENV:Header>
   <ns:elem xmlns:ns="http://www.ns_uri.com"
    soap:role="http://www.w3.org/2001/12/soap-envelope/role/next">
   </ns:elem>
</SOAP-ENV:Header>
```

Zusätzlich zu diesen drei standardmäßigen Rollennamen können für Anforderungen von Applikationen noch andere spezifisch definiert werden. Eine Rolle dient lediglich dazu, einen Knoten zu identifizieren, wobei sie jedoch keine semantische Beschreibung des Routings oder des Nachrichtenaustausches beinhaltet. So kann z.B. eine Rolle durch einen URI definiert sein, zu dem eine SOAP-Nachricht geroutet werden soll. Es ist dagegen auch möglich, Rollen nur indirekt an das Routing zu binden und eher die Funktion des Knotens zu beschreiben, z.B. mittels des URI „http://www.fh-sbg.ac.at/AccountManager“.

Rollen werden in einer SOAP-Nachricht optional im Header mittels eines *role*-Attributes definiert. Eine Nachricht hat also einen Knoten als Ziel, wenn die Rolle, die im Header-Block beschrieben ist, eine Rolle ist, in der der Knoten operiert.

Header-Blöcke können nur dann verwendet werden, wenn der Empfänger die enthaltene Information auch zur Gänze versteht. Wenn der Header ein *mustUnderstand*-Attribut beinhaltet, das auf den Wert *true* gesetzt ist, muss der Block obligatorisch vorhanden sein. Wie das Attribut syntaktisch in den SOAP-Header eingebaut wird, ist in [GUDG03a] beschrieben.

Die eigentliche Verarbeitung und Weiterleitung von SOAP-Nachrichten geschieht in folgenden Schritten:

- Eruieren aller Rollen, in denen der Knoten operiert
- Identifizierung aller Header-Blöcke, die den Knoten als Ziel haben und verpflichtend sind
- Generierung einer Fehlermeldung, wenn ein oder mehrere Header-Blöcke nicht vollständig verstanden wurden; wenn solch ein Fehler auftritt, wird die Nachricht nicht weiter bearbeitet
- Verarbeitung aller Header-Blöcke, die an den Knoten adressiert sind und wenn dieser der Endempfänger ist, wird auch der Körper der Nachricht verarbeitet
- Wenn der Knoten nur eine weiterleitende Funktion hat, wird die Nachricht weitergesendet. Dieser Vorgang wird in [GUDG03a] näher erläutert.

2.3.3 Erweiterbarkeitsmodell

Ein SOAP-Feature ist eine Erweiterung des Datenaustausches mittels SOAP, die es ermöglicht, den Datentransfer selbst noch mit bestimmten Fähigkeiten (im Folgenden auch als Merkmale, Besonderheiten oder Eigenschaften bezeichnet) auszustatten. Solchen Besonderheiten sind zwar keine Grenzen gesetzt, es gibt jedoch einige beispielhafte Implementierungen wie Verlässlichkeit, Sicherheit, Wechselbeziehungen, Routing oder so genannte Message Exchange Patterns (MEPs), die ein Muster für die Kommunikation zwischen zwei Knoten festlegen, wie besipeilsweise Request-Response, One-Way oder Peer-to-peer Kommunikation.
SOAP bietet zwei Mechanismen, mittels derer Features verwendet werden können; erstens das in Unterkapitel 2.3.1 beschriebene Prozessmodell, das das Verhalten bzw. die Rolle eines Knotens beschreibt und zweitens die SOAP Einbindungsvorgaben, die festlegen, wie Nachrichten über ein tiefer liegendes Protokoll gesendet und empfangen werden.
Bei der Verwendung von End-to-end Features soll danach getrachtet werden, dass sie als SOAP Header-Blöcke eingesetzt werden, damit Regeln und Rollen des Prozessmodells eingesetzt werden können.
Grundsätzlich muss jedes Merkmal Folgendes beinhalten:

- Einen URI, der den Namen der Fähigkeit definiert
- Die Information, die bei jedem Knoten benötigt wird, um das Feature ausführen zu können
- Die notwendige Verarbeitung für jeden Knoten, um alle Vorgaben der Besonderheit zu erfüllen, was z.B. auch Fehlerbehandlung inkludiert
- Die Information, die der Knoten übertragen soll

Eine weitere Möglichkeit, Features zu implementieren, sind SOAP Module, die die Syntax eines Header-Blocks definieren. Eine detaillierte Beschreibung dieser Module kann in [GUDG03a] gefunden werden.

2.3.4 Einbindungsrahmenwerk

Nachdem SOAP auf Basis einer Vielzahl von darunter liegenden Protokollen funktioniert, bedarf es einer formalen Regelung, wie diese Protokolle eingebunden werden müssen. Ein solches Rahmenwerk erfüllt folgende Aufgaben:

- Deklaration der verfügbaren Features
- Beschreibung, wie Services von darunter liegenden Protokollen verwendet werden, um SOAP-Informationen zu übertragen und verfügbare Features einzubinden
- Beschreibung der Fehlerbehandlung
- Definition der Anforderungen für die Implementierung der Einbindung

Diese Einbindung ist Teil von SOAP, ist somit kein Knoten und besitzt kein separates Prozessmodell. Die Ziele des Rahmenwerkes sind erstens alle gemeinsamen Anforderungen und Konzepte zusammenzufassen, zweitens ein homogene Beschreibung von Situationen zu erleichtern, in denen mehrere Einbindungen gemeinsame Features unterstützen und letztlich um einfacher konsistente Spezifikationen von optionalen Features kreieren zu können.

Die gesamte Protokollspezifikation sowie übersichtliche Codebeispiele von SOAP werden in [BOX00] dokumentiert.

2.4 Web-Services

Ein Web-Service wird durch einen URI identifiziert, dessen öffentliche Schnittstellen mit XML beschrieben sind. Um die Funktionen des Service nutzen zu können, müssen die Schnittstellen in bestimmter Weise angesprochen werden, wobei der Datenaustausch mit XML und über internetbasierte Protokolle erfolgt.
Der Anfang von Web-Services findet sich in einem Protokoll, das wie der Name *RPC over HTTP via XML* einen entfernten Methodenaufruf über HTTP mit XML erlaubte. Diesem 1998 entstandenen Protokoll fehlte jedoch eine aussagekräftige Meta-Beschreibung, was es notwendig machte, ein Beispieldokument für einen Methodenaufruf zur Verfügung zu stellen. Dies wiederum limitierte gleichzeitig den Umfang und den Einsatzbereich des Protokolls, die Grundidee für ein Web-Service war prinzipiell jedoch geschaffen. Mehr Informationen über RPC over http via XML können in [WINE98] gefunden werden.

Anwendungen, die über einen so genannten Thin-Client[3] ablaufen, erleben zur Zeit einen starken Aufschwung, da es mit Hilfe dieser Applikationen sehr einfach ist, bestimmte Informationen und Services zu verbreiten. Wenn auf einem Web-Server Informationen durch pures HTML zur Verfügung gestellt werden, so kann dies nur in statischer Weise geschehen, es wird also ein fester Inhalt zur Verfügung gestellt. Dynamische Antworten auf individuelle Anfragen erfordern dynamische Komponenten im Web-Server. In den Server integrierte Programme sind z.B. Perl- oder Common Gateway Interface (CGI) Module. Weit verbreitete Java-Erweiterungen für Web-Server sind Servlets, welche z.B. von HTTP-Anfragen aufgerufen werden und darauf hin eine Ausgabe erzeugen. Eine nähere Beschreibung von Servlets folgt in Unterkapitel 2.7.

2.4.1 Eigenschaften von Web-Services

Web-Services zeichnen sich durch diverse Eigenschaften aus, siehe z.B. [HEIN03]. Diese Charakteristika werden im Folgenden kurz erläutert.

Lockere Bindung

Diese Eigenschaft bedeutet, dass es keine speziellen Benutzerdaten braucht, um ein Web-Service nutzen zu können, was in der Regel durch Eingabe eines URL geschieht. Prinzipiell kann jeder das Service in Anspruch nehmen, der Zugang dazu hat. Der Vorteil dieses Systems ist, dass sich der Server keine unnötigen Daten über den Nutzer merken muss, ein Nachteil ergibt sich jedoch aus der Tatsache, dass Links auf ein Web-Service ins Leere verweisen, wenn dieses entfernt wird. Deshalb wurde die unverbindliche Vorgabe für Designer von Web-Services aufgestellt, dass ein Service nicht mehr entfernt werden darf, sobald der URL einmal veröffentlicht worden ist.

Extensible Markup Language als Basis

Ursprünglich wurde HTML als Grundlage für Web-Services verwendet, was aber, wie oben erwähnt, Probleme bei dynamischer Antwortgenerierung bereitet. Deshalb wird momentan XML als Grundlage propagiert, um die Vorteile einer standardisierten Datenaustausch-möglichkeit und einer breiten Unterstützung von diversen Programmen zu nützen.

---

[3] Möglichst schlank gebauter Client in einer Client-Server Struktur; dieser Ansatz überlässt komplexe Berechnungen dem Server

Synchronität und Asynchronität

Bei der synchronen Verwendung von Web-Services wird nach absetzen der Anfrage so lange gewartet, bis eine Antwort retourniert wurde. Dies wird bei Services angewandt, bei denen eine Ausgabe auf eine Eingabe hin erwartet wird.
Wird ein Web-Service asynchron betrieben, wird die Anfrage gestellt und unmittelbar danach mit anderen Prozessen fortgefahren. Gegebenenfalls können eine oder mehrere Antworten vom Service auch später erfolgen.

Unterstützung von Remote Procedure Calls

Ein Remote Procedure Call (RPC) ist ein Funktionsaufruf auf einem entfernten Rechner. Diese sind bei Web-Services üblicherweise synchron implementiert.

Dokumentaustausch

Neben dem Austausch von Nachrichten, die das Web-Service anbietet, können auch applikationsspezifische XML-Dokumente ausgetauscht werden.

Integration

Durch die breite Unterstützung, die standardisierte Basisprotokolle wie dem Hyper Text Transfer Protocol (HTTP) oder dem Simple Mail Transfer Protocol (SMTP) als auch der standardisierte Datenaustausch via XML bieten, erlaubt eine relativ simple Integration verschiedenster Systeme.

2.4.2 Struktur eines Web-Service

Die folgende Abbildung illustriert den generellen Aufbau eines Web-Service mit den darin enthaltenen Technologien. Wie aus Abbildung 2.3 ersichtlich bauen Web-Services auf einer Reihe von standardisierten Technologien auf.

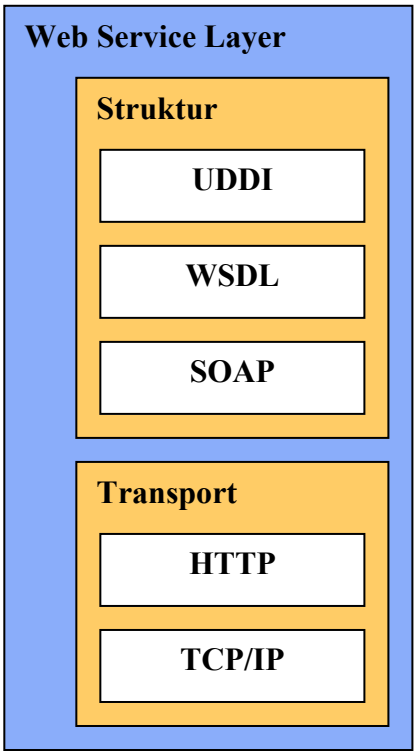

Abbildung 2.3: Schichten eines Web-Service, nach [HEIN03].

Der Web Service Layer besteht strukturell aus UDDI zur Identifizierung, WSDL – in Abschnitt 2.4.5 detailliert beschrieben - zur Beschreibung des Service und SOAP als Übertragungsprotokoll. Der Datentransport selbst wird mittels HTTP auf TCP/IP realisiert.

2.4.3 Java und .NET Web-Services

Die Unterscheidung von Web-Services, die auf Java bzw. .NET basieren, soll an dieser Stelle erläutert werden, um die Interoperabilität zwischen Windows- und Unix-Systemen zu veranschaulichen. Web-Services könnten in den kommenden Jahren improvisierte Datenübertragungsmechanismen wie z.B. Datenbankimporte oder Dateidumps ersetzen.

.NET

Das Problem der Windows-internen Architektur liegt darin, dass Microsoft-Komponenten zwar sehr leicht integriert werden können, die kometenhafte Entwicklung des Internet jedoch Standardprotokolle wie Common Object Request Broker Architecture (CORBA) oder HTTP hervorbrachte und Windows damit zwang, auch mit diesen interoperabel zu sein. Dadurch mussten neue Schnittstellen geschaffen werden, um auch über die Grenzen von Windows-basierten Netzwerken kommunizieren zu können.
Auf Grund dieser Schwächen wurde schlussendlich ein völlig neues Framework aufgebaut, das eine durchaus ernstzunehmende Konkurrenz zu bereits etablierten Technologien darstellt:

Microsoft .NET. Das Grundkonzept von .NET beruht auf dem Austausch von Software zwischen Computern jeglicher Art. Der Software Development Kit (SDK) dient als Basis für die Softwareentwicklung mit .NET und unterstützt unter anderem auch die Generierung von Funktionen für Web-Services.
Die Struktur einer mit .NET erstellten Web-Service-Applikation ist in der folgenden Abbildung dargestellt:

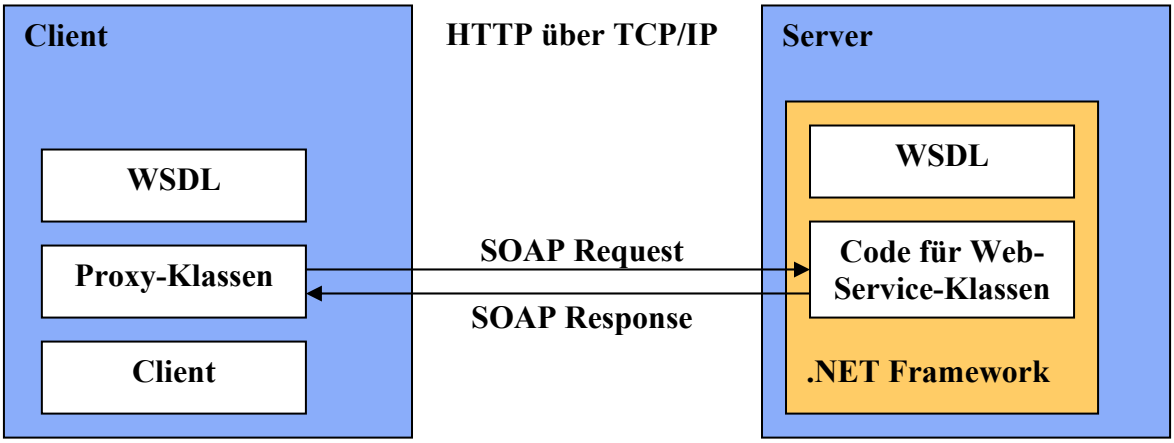

Abbildung 2.4: Struktur eines Web-Service mit .NET, nach [HEIN03].

Der Client in einem .NET-Web-Service besteht aus einem WSDL-Modul, das auf den Proxy-Klassen aufsetzt, über die der SOAP-Nachrichtenaustausch stattfindet; darunter liegt das Clientprogramm selbst. Im Server sind die Servicebeschreibung und der gesamte Code für Web-Service-Klassen in das .NET Framework eingebunden. Der Datentransport selbst geschieht wie bereits in Abschnitt 2.4.2 erwähnt, mittels HTTP auf TCP/IP.

2.4.4 Frameworks für Java Web-Services

Die Entscheidung für Java als Basis des in diesem Projekt kreierten Web Service fiel auf Grund der Existenz von Programmierschnittstellen (APIs) dieser plattformunabhängigen Programmiersprache für jede darunter liegende Technologie. Speziell für die Verwendung in Verbindung mit XML und SOAP zeichnet sich Java durch eine breite Unterstützung und als viel verwendete Web-Service-Basis als bewährte Technologie aus.
Ein Framework ist ein Bündel von verschiedenen Technologien, um damit ein geschlossenes Produkt für applikationsspezifische Anforderungen zu bieten. Für Web-Services bestehen mehrere Frameworks, wobei an dieser Stelle das Apache Extensible Interaction System (AXIS), der IBM Web Service Toolkit und der Java Web Services Developer Pack näher behandelt werden.

Apache AXIS

AXIS ist eine Weiterentwicklung von Apache SOAP, das einige Mankos gegenüber dem Vorgänger ausmerzt, so wie z.B. mangelnde Unterstützung von WSDL, kein Zugriff auf den SOAP-Header bei RPC-Implementierung oder ungenügende Unterstützung von anderen Transportprotokollen als HTTP.
Apache AXIS wurde mit dem Ziel implementiert, eine möglichst modulare und erweiterbare Struktur zu schaffen bei voller Unterstützung von SOAP Version 1.2. AXIS stellt mit dem Web Service Deployment Descriptor (WSDD) einen sehr effizienten Deployment-mechanismus zur Verfügung, der natürlich vorkonfiguriert werden muss. Außerdem unterstützt AXIS ein Transport Framework, wodurch das untergeordnete Übertragungs-protokoll lediglich eine nebensächliche Rolle für die Funktion des Web-Service spielt.
Voraussetzung für die Installation von AXIS sind der Java Development Kit (JDK) ab Version 1.4, Jakarta Tomcat ab Version 4.0.3, der Xerces Parser sowie das Java Activation Framework (JAF).

IBM Java Web Service Tool Kit

Der IBM Java Web Service Tool Kit (JWSTK) umfasst prinzipiell die selben Pakete wie der Java Web Services Developer Pack von SUN, beinhaltet jedoch zusätzlich noch den Java WSDP Registry Server, einen einfachen UDDI-Server.

Der Java Web Services Developer Pack (JWSDP) wird in Unterkapitel 2.5.1 gesondert behandelt.

2.4.5 Web Service Description Language

Die Web-Service Description Language (WSDL) dient, wie der Name schon vermuten lässt, dazu, ein Web-Service zu beschreiben. In einem XML-Dokument wird festgehalten, wo sich das Service befindet, welche Methoden es zur Verfügung stellt und wie diese angesprochen werden können. Ein WSDL-File setzt sich aus folgenden Elementen zusammen:

- **Types** – Datentypdefinitionen mit Hilfe eines „Typsystems“ (z.B. XSD)
- **Message** – abstrakte Definition der ausgetauschten Daten
- **Operation** – Beschreibung einer vom Web-Service unterstützen Methode
- **Port Type** – Menge von Operationen, die von Endpoints unterstützt werden
- **Binding** – Protokoll und Datenformatspezifikation für einen *Port Type*
- **Port** – einzelner Endpoint (Kombination von Bindings und einer Netzwerkadresse)
- **Service** – Menge verwandter Endpoints

Mehr Informationen über WSDL können in [CHRI01] und [REFS05a] gefunden werden.

2.4.6 Universal Description, Discovery and Integration

Universal Description, Discovery and Integration (UDDI) ist ein über SOAP kommunizierender Verzeichnisdienst, über den Unternehmen Web-Services registrieren und suchen können. In UDDI-Verzeichnissen können durch WSDL beschriebene Web-Service Schnittstellen und andere Informationen deponiert werden. Es besteht sowohl die Möglichkeit, eine Web-Service Beschreibung auf einem zentralen UDDI-Server von IBM, Microsoft und anderen Großkonzernen zu positionieren, oder aber seinen eigenen Server einzurichten. Eine detaillierte Beschreibung von UDDI wird in [UDDI00] gegeben.

2.5 Java

Bezüglich dieser objektorientierten Programmiersprache soll an dieser Stelle nur festgehalten werden, dass sie prädestiniert für den Einsatz für Web-Services ist. Durch die vordefinierten Klassen und gut strukturierte Dokumentation der Application Programming Interfaces (APIs) eignet sich die Java am besten für das Projekt *adr_soap*. In [SUN05a] wird eine genaue Spezifikation der Syntax und der APIs dargeboten.

2.5.1 Java Web Services Developer Pack

Der Java Web Services Developer Pack (JWSDP) ist ein kostenloser Toolkit, mit dem XML-Applikationen, Web-Services und Web- Applikationen erstellt und getestet werden können. Der Toolkit enthält folgende Pakete:

- Apache Ant (Kompilierung von Softwarepaketen)
- Apache Tomcat (Web-Server, Servlet-Container, s. Abschnitt 2.9)
- JAXM (Java API for XML Messaging)
- JAXB (Java Architecture for XML Binding)
- JAXP (Java API for XML Processing)
- JAXR (Java API for XML Registry)
- JAX-RPC (Java API for XML-based RPC)
- Java Server Pages Standard Tag Library (STL)
- SAAJ (SOAP with Attachments API for Java)

Auf die Details der Verwendung dieser APIs wird im Bericht im Zuge der technischen Implementierungsbeschreibung näher eingegangen. Mehr über die APIs und den JWSDP kann in [HEIN03] und [SUN05c] gefunden werden.

2.5.2 Eclipse

Als Java-Entwicklungsumgebung wurde Eclipse ausgewählt. Gründe dafür sind freie Verfügbarkeit und modulare und individuelle Einsatzmöglichkeiten, die durch diverse Plug-Ins zusätzlich zur Grundfunktionalität gegeben sind. Eclipse wurde als universelles Programmiertool entwickelt. Es unterstützt die Bearbeitung von Quellcode aus mehreren Programmiersprachen, darunter HTML, Java, C, JSP, XML etc. Eclipse steht kostenlos zum Download zur Verfügung, das Copyright des Source-Codes hält jedoch IBM. Mehr Informationen über die Eclipse Foundation und alle Features des Programms können auf [ECLI04] und in [ASSI04] gefunden werden.

2.6 Java Server Pages

Java Server Pages (JSP) ermöglichen die Erstellung von Web-Services, die sowohl statische als auch dynamische Elemente enthalten. Generell kann JSP als eine Verbindung von textbasierten Auszeichnungssprachen mit Java-Quellcode gesehen werden. So wird in HTML zum Beispiel Java-Code mittels <% … %> direkt in die Seite eingebunden. Mehr Informationen über die JSP-Technologie werden in [AYER99] und auf [SUN05b] zur Verfügung gestellt.

2.7 Servlets

Ein Servlet ist eine Java-Klasse, die sich auf einem Web-Server befindet und es anderen Rechnern ermöglicht, auf ein Web-Service mittels eines Request-Response-Modells zuzugreifen. Ursprünglich wurden Servlets entwickelt, um HTTP-Requests zu bearbeiten. Mittlerweilen können viele Typen von Anfragen verarbeitet werden, unter anderem auch SOAP-Nachrichten.

Servlet Containers, auch Servlet Engines genannt, dienen dazu, Anfragen aufzunehmen, diese vom darunter liegenden Protokoll in Objekte zu übersetzen und die vom Servlet generierte Antwort wieder zurückzuschicken. Das Interface zwischen Servlet und Container wird durch die Servlet API spezifiziert. Der Servlet Container, der in diesem Projekt verwendet wird, wird implizit vom Tomcat Web-Server zur Verfügung gestellt.

Für dieses Projekt sind Servlets essentiell, weil die Datenbankabfrage, die Ansteuerung des Koordinatentransformationsservices, die Verarbeitung des eingegebenen Adressstrings oder auch das Parsen der XML-Konfigurationsdatei einen zu hohen Rechenaufwand für eine clientseitige Verarbeitung bedeuten würden.

Weitere Funktionen, Implementierungsdetails, verschiedene Arten und Beispiele bezüglich Servlets werden in [AYER99] und [BODO04] beschrieben.

2.8 Java Database Connectivity

Java DataBase Connectivity (JDBC) definiert eine Datenbankschnittstelle für Java basierend auf dem CLI-Standard.. Die JDBC-relevanten Klassen befinden sich im package *java.sql.**. Außerdem müssen spezifische Bibliotheken für die jeweiligen Datenbanken einbezogen werden.
Prinzipiell besteht ein JDBC-Programmablauf aus sechs Stufen:

- Import der JDBC-Klassen
- Verbindungsaufbau zur Datenbank
- Datenbankzugriff via SQL
- Datenverarbeitung mittels eines Cursors
- Exception Handling
- Verbindungsabbau

Ein JDBC Tutorial steht unter [FISH04] zur Verfügung:

2.9 Apache Tomcat

Tomcat ist ein Web-Server, der als Open-Source-Projekt von *The Apache Software Foundation* entwickelt wurde. Als kostenloser und übersichtlicher Web-Server eignet sich Tomcat ideal für die Verwendung in diesem Projekt. Eine detaillierte Beschreibung kann auf [APAC04] und in [MOCZ05] gefunden werden. Die Konfiguration und Eigenschaften des Servers werden in den Kapiteln 5.3.2 und 5.4.2 näher behandelt.

3 Grundstruktur des Web-Service *adr_soap*

Die generelle Funktionsweise des Adressen-Web-Service ist in der folgenden Abbildung illustriert:

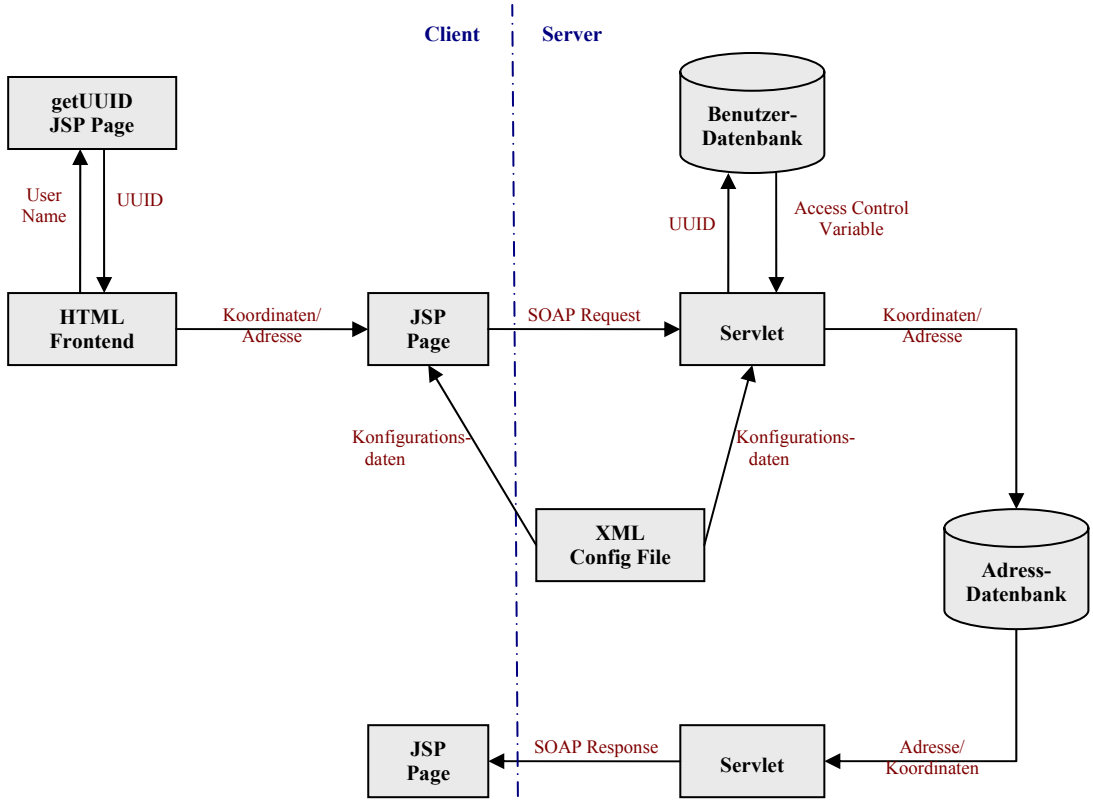

Abbildung 3.1: Ablauf der Adressabfrage.

In das HTML-Frontend wird ein Benutzername eingegeben, aus dem dann über eine JSP-Seite (*getUUID*) eine UUID generiert wird. Danach werden in ein Formular im Frontend Koordinaten oder eine Adresse eingegeben, die dann an eine JSP-Seite (*SOAP_Request*) weitergegeben werden. Diese verpackt die Eingabedaten in einen standardisierten SOAP-Request und schickt ihn zum Servlet, das die Daten wieder herausfiltert und damit die Datenbankabfrage durchführt, nachdem überprüft wurde, ob der Benutzer für diese Abfrage authorisiert ist. Die Ergebnisdaten werden wieder in eine SOAP-Nachricht eingebaut und zurück an die JSP-Page geschickt, die das wiederum ausgepackte Resultat ausgibt.

Diese Basisstruktur wird von zahlreichen zusätzlichen Funktionen ergänzt, die im folgenden Kapitel aufgelistet werden.

4 Liste der Applikationsfeatures

Diese Liste beinhaltet die implementieren Web-Service-Features in sechs Kategorien. Die in diesem Abschnitt aufgelisteten Features werden im Zuge der Beschreibung der technischen Implementierung in Kapitel 5 und 6 detailliert behandelt.

Basisservlet (SOAP)

- Koordinaten- bzw. Adresssuche
- Kommunikation über SOAP
- Dynamische Datenbankabfrage

Um die Grundfunktionalität des implementierten Web-Service zu erweitern und die Anwendung somit zu einem praktikablen Prototyp zu machen, sind folgende Implementierungen im Projekt enthalten:

XML Config File

- XML-Konfigurationsdatei für Parametervoreinstellung (Datenbankzugriff, Pufferwerte für die Umkreissuche, verwendetes Koordinatensystem, etc.)

Sicherheit

- Universal Unique Identifier (UUID) zur Benutzerauthentifizierung
- Verwendung eines Session-Attributes, um den fehlerlosen Informationsfluss zu gewährleisten

Datenbanken

- Unterstützung aller gängigen Datenbanken
- Dynamische Generierung des Select-Statements
- Rückgabe von maximal 100 Resultaten zur Performance-Steigerung

Koordinaten & Transformation

- Standardisierte Koordinatenrückgabe in GML
- Ansteuerung eines Koordinatentransformationsservices zur zusätzlichen Rückgabe in WGS84-Darstellung

Applikationsinterne Features

- Fehlerbehandlung bei ungenauer Adresseingabe

  JSP-Seite

  - Dynamische Aufteilung der Eingabe im Adressfeld in Straße, Hausnummer und deren alphanumerischen Zusatz
  - Umwandlung diverser Zeichenketten (z.B. „str “, „str.“, etc. zu „straße“) zwecks Konformität der Eingabe mit Datenbankeinträgen

  Servlet

  - Halbierung des Adress-Strings bei leerer Ergebnismenge nach der Datenbankabfrage
  - keine Unterscheidung zwischen Klein- und Großschreibung
- Umkreissuche bei ungenauer Koordinateneingabe je nach Datenbank:
  - Umkreissuche mit Oracle Spatial
  - Umgebungssuche im Rechteck
- Exemplarische Ausgabe des Ergebnisses durch Verlinkung zur DORIS-Karte
- Verpackung diverser Abfragen in Funktionen, was eine modulare Erweiterbarkeit sicherstellt

Klientseitige Konfiguration (HTML & JSP)

- Eingabe der Umkreisgröße mit Hilfe eines Schiebereglers
- Möglichkeit, eine Abfrage in mehreren Datenbanken abzusetzen
- Dynamisches Auslesen der Server-URL, auf dem die Anwendung liegt → NUR das Konfigurations-File muss verändert werden

5 Technische Beschreibung der Implementierung

Dieser Abschnitt enthält eine detaillierte Beschreibung der Implementierung und der Aufgaben der in Abbildung 3.1 dargestellten Module.

5.1 Konfigurationsdatei

Aus dem Config-File werden sowohl von der JSP-Seite als auch vom Servlet diverse Parameter ausgelesen. Diese enthalten Angaben über die Verbindung zur Datenbank, das verwendete Koordinatensystem, die Spaltennamen der auszulesenden Elemente in der Adresstabelle sowie einige Einstellungen, die für die Umkreissuche (nötig bei ungenauer Koordinateneingabe) relevant sind. Die vier Basiselemente der Konfigurationsdatei sind:

- *buffer* (Werte für Umkreissuche)
- *database* (Parameter für die Verbindung zur Adressdatenbank)
- *security_db* (Parameter für die Verbindung zur Security-Datenbank)
- *units* (verwendetes Koordinatensystem)

Eine detaillierte Beschreibung aller einzelnen Elemente wird in Kapitel 7 (Benutzerinformationen) gegeben.

5.2 HTML-Frontend

Das HTML-Frontend beinhaltet ein Formular (`method=post`), in das Daten auf zwei Arten eingegeben werden können. Erstens können x-, y- und z-Koordinaten dazu dienen, eine übereinstimmende Adresse zu finden, oder aber es wird eine Adresse optional mit Hausnummer und deren alphanumerischen Zusatz, eventuell eine Gemeinde und eine Postleitzahl eingegeben, um die dazugehörigen Koordinaten abzufragen. Bei der Suche mit Koordinateneingabe kann überdies der Radius für eine eventuelle Umkreissuche mit Hilfe eines Schiebereglers eingestellt werden.

Ein Screenshot des Frontends wird in der folgenden Abbildung illustriert:

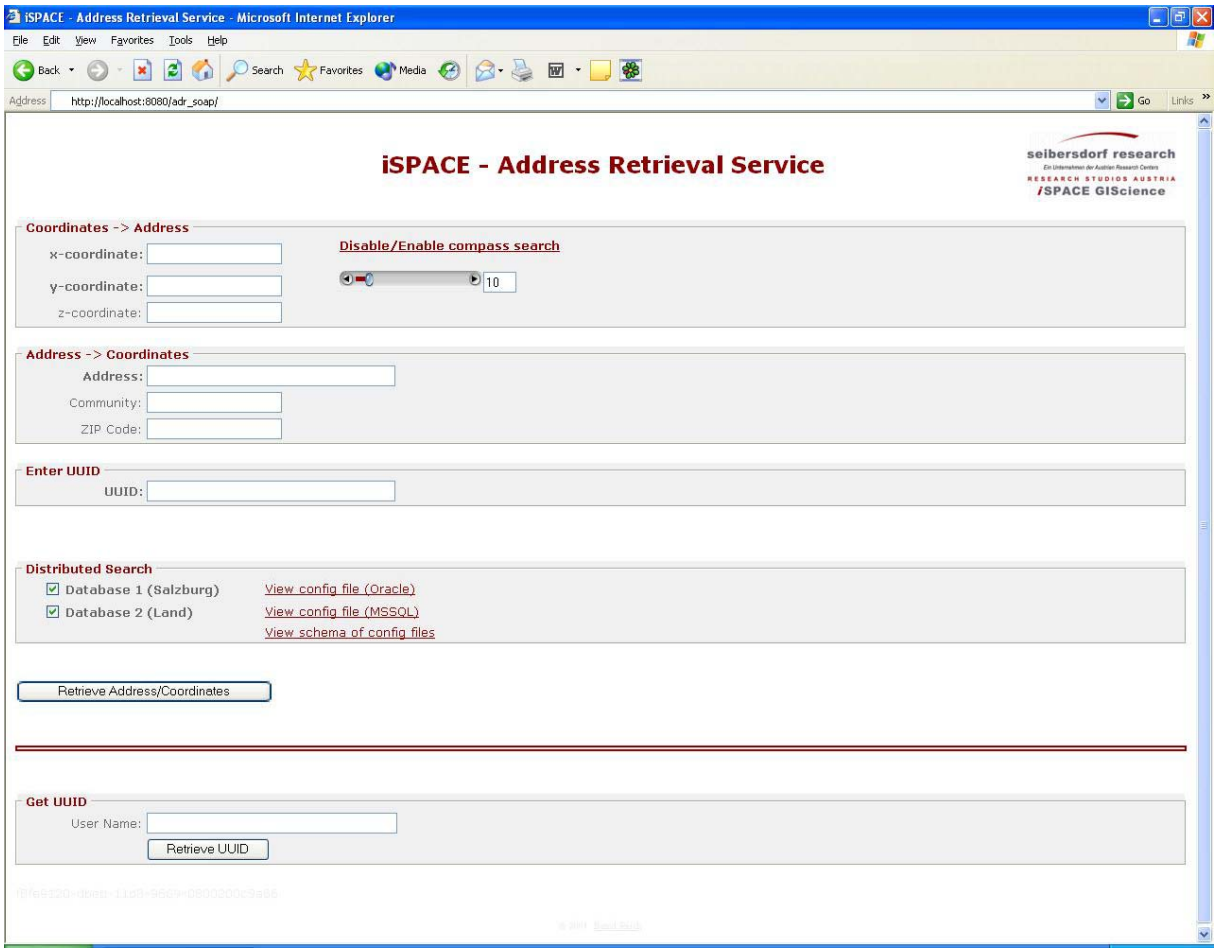

Abbildung 5.1: Screenshot des HTML Frontends.

Das Eingabefeld, in welches eine UUID eingegeben wird, dient zur Benutzer-authentifizierung. Die folgenden zwei Checkboxen, die optional gesetzt werden können, definieren, in welchen Datenbanken die Suche durchgeführt werden soll.
Im unteren Abschnitt der Homepage befindet sich noch ein Eingabefeld, über das durch Eingabe des eindeutigen Benutzernamens mit Hilfe eines weiteren JSP-Programms die entsprechende UUID generiert wird.

Zusammengefasst bietet das HTML-Frontend also folgende Eingabefunktionen:

- Adresssuche mit Koordinateneingabe (mit Umkreissuche)
- Koordinatensuche mit Adresseingabe (mit Fehlerbehandlung der Eingabe)
- Eingabe der Umkreisgröße mit Hilfe eines Schiebereglers
- Generierung einer UUID mittels einer eigenen JSP-Seite
- Auswahl der Datenbanken durch Checkboxenwerte

5.3 JSP-Seite

Im Projekt werden wie oben bereits erwähnt zwei JSP-Seiten verwendet:

- getUUID.jsp
- SOAP_Request.jsp

Die erste generiert eine UUID aus einem eingegebenen Benutzernamen. Die entscheidenden Java-Befehle für diesen Vorgang sind:

```
UUIDGenerator gen = UUIDGenerator.getInstance();
UUID user_id = gen.generateNameBasedUUID(null, ws_user);
```

Hierfür müssen die entsprechenden Klassen `org.doomdark.uuid.*;` importiert werden, es muss jedoch keine zusätzliche Klassenbibliothek eingebunden werden.

Die zweite JSP-Page dient als Interface zwischen der HTML-Seite und dem Servlet. Nach dem Auslesen des Konfigurations-Files nimmt es die Eingabedaten vom Frontend entgegen, baut daraus einen SOAP-Request und schickt diesen an das Servlet. Sie ist ungleich komplexer aufgebaut als die erste JSP-Seite und wird deshalb in der Folge detailliert beschrieben.

5.3.1 Implementierung

Generell wurde bei der Implementierung, sowohl der JSP-Seiten als auch der Servlets, darauf geachtet, möglichst viele Abfragen in Funktionen zu verpacken, um eine modulare Erweiterbarkeit zu gewährleisten.

Die oben genannten Eingabedaten können sowohl aus Koordinaten als auch einer Adresse bestehen. Sie werden – genauso wie die Variablen für den Slider-Wert (Umkreissuche), die UUID und die Checkboxenwerte für die abzufragenden Datenbanken - mittels des Befehls

```
request.getParameter(Variablenname)
```

von der JSP-Seite entgegengenommen. Bevor die SOAP-Nachricht gebaut wird, werden noch einige Korrekturen der eingegebenen Adressdaten durchgeführt:

- „str “, „str.“, „strasse“ → „straße“
- Aufschlüsselung des Adress-Strings in drei Werte (Straße, Nummer, alphanumerischer Zusatz)

Als erstes werden verschiedene Schreibweisen des Wortes „straße“ in diesen standardisierten String konvertiert. Eine Funktion sucht statisch nach den Zeichenfolgen „str “, „str.“ und „strasse“ und wandelt diese um.

In weiterer Folge wird der eingegebene String, in die drei Elemente aufgeschlüsselt, die er enthält. Ein wichtiges Faktum für die praktische Anwendung ist, dass der String nicht in einer eindeutigen bestimmten Form eingegeben werden muss, sondern dynamisch die einzelnen Komponenten herausgefiltert werden.

Um diese Eingabeinformationen wird dann eine SOAP-Nachricht, mit ihren Bestandteilen *Envelope*, *Header* und *Body*, gebaut, was durch die folgenden Codezeilen, die die Nachrichtenerstellung bei Koordinateneingabe dokumentiert, geschieht.

- Zuerst müssen im Initialisierungsteil Instanzen für die Message selbst erstellt werden:

```
SOAPConnectionFactory soapConnectionFactory =
 SOAPConnectionFactory.newInstance();
SOAPConnection connection = soapConnectionFactory.createConnection();
SOAPFactory soapFactory = SOAPFactory.newInstance();
MessageFactory factory = MessageFactory.newInstance();
```

- Hierauf wird eine leere SOAP-Nachricht erstellt.

```
SOAPMessage message = factory.createMessage();
SOAPHeader header = message.getSOAPHeader();
```

- Anschließend werden dem SOAP-Header dir Attribute für die UUID (s. folgender Codeabschnitt) und die Session ID hinzugefügt:

```
Name un = soapFactory.createName("uuid");
header.addAttribute(un, _UUID);
```

- Darauf hin wird der eigentliche Körper der Nachricht geladen, ein *Name*-Objekt erstellt und ein Element hinzugefügt.

```
SOAPBody body = message.getSOAPBody();
Name bodyName = soapFactory.createName("getAddress", "coords",
 "http://mssql.researchstudio.at:8081");
SOAPBodyElement getaddress = body.addBodyElement(bodyName);
```

- Als nächster Schritt wird die Elementstruktur der SOAP-Message festgelegt. Im folgenden Quelltextauszug wird eine GML-konforme Darstellung eines Punktes erstellt. Die Übergabeparameter der Funktion `createName()` bedeuten, dass der Namespace *gml* auf dem Server *http://www.opengis.net/gml* liegen und das Element *location* enthalten muss. Die leere Message wird dann mit den aus dem Eingabeformular erhaltenen Werten befüllt. Als letztes werden dem *point*-Element die für die GML-Darstellung notwendigen Attribute mitgegeben, was mit Hilfe der letzten hier aufgelisteten Codezeile zu sehen ist.

```
Name loc = soapFactory.createName("location", "gml",
 "http://www.opengis.net/gml");
Name poi = soapFactory.createName("Point", "gml",
 "http://www.opengis.net/gml");
 Name pos = soapFactory.createName("pos", "gml",
 "http://www.opengis.net/gml");
Name srs = soapFactory.createName("srsName");
SOAPElement lo = getAddress.addChildElement(loc);
SOAPElement point = lo.addChildElement(poi);
SOAPElement posi = point.addChildElement(pos);
posi.addTextNode(_xcoord + " " + _ycoord + " " + _zcoord);
point.addAttribute(srs, (owner + ":" + Long.toString(EPSG_ID)));
```

- Die Darstellung des Punktes in GML sieht dann also folgendermaßen aus:

```
<gml:location xmlns:gml="http://www.opengis.net/gml">
   <gml:Point srsName="EPSG:31289">
      <gml:pos>x y z</gml:pos>
   </gml:Point>
</gml:location>
```

- Diese SOAP-Message wird dann an das Servlet geschickt, wobei die Variable *endpoint* den Bezeichner des Web-Service und den Aliasnamen des Servlets enhalten muss:

```
URL endpoint = new URL(server_url + "adr_soap/RequestHandler");
SOAPMessage response = con.call(message, endpoint);
```

- Danach wartet die JSP-Seite auf eine Antwort. Bei Erhalt dieser wird die Verbindung mit dem Befehl `connection.close();` abgebaut.

- Anschließend wird die SOAP-Response geparst und die entsprechenden Elemente werden, nach Überprüfung einer Control-Variable zur Authentifizierung, iterativ ausgegeben.

5.3.2 Technische Voraussetzungen

Einige technische Installationen sind notwendig, um die oben genannten Operationen durchzuführen. Erstens muss ein Java Web-Server installiert werden, auf dem die Webseiten platziert werden können. Der in Kapitel 2.9 erwähnte Tomcat Server beinhaltet eine sehr gute Dokumentation und ist deshalb relativ einfach zu konfigurieren. Bei der Erstellung dieses Web-Service wurde die Version 5.0.19 verwendet.
Webapplikationen müssen im Home-Verzeichnis des Servers in den Ordner *webapps* gelegt werden. Darin befindet sich ein Verzeichnis namens *WEB-INF/lib*, in dem alle jar-Dateien abgelegt werden, die für die eigene Applikation notwendig sind. Diese können jedoch auch im Homeverzeichnis des Servers in den Ordnern *common/lib* oder *shared/lib* platziert werden.

In ersterem Fall sind die Bibliotheken für alle Webapplikationen und für den internen Tomcat Code sichtbar. Wenn sich die jar-Files in letzterem Verzeichnis befinden, sind sie nur für die Webanwendungen sichtbar.

Diese oben genannte Ordnerstruktur wird durch eine wichtige Datei hergestellt, die der Vollständigkeit halber erwähnt werden muss, nämlich *WEB-INF/web.xml*, auch der „Web Application Deployment Descriptor“ genannt. Dieses XML-File, das dazu dient, Servlets und andere Komponenten der Webapplikation zu beschreiben wird in Abschnitt 6.1.1 näher behandelt.

Der Übersichtlichkeit halber wird in der folgenden Grafik die gesamte für den Tomcat Web-Server relevante Verzeichnisstruktur abgebildet. Darin enthalten ist auch die in Kapitel 6.1.1 dokumentierte Ordnergliederung des Web-Application-Archives.

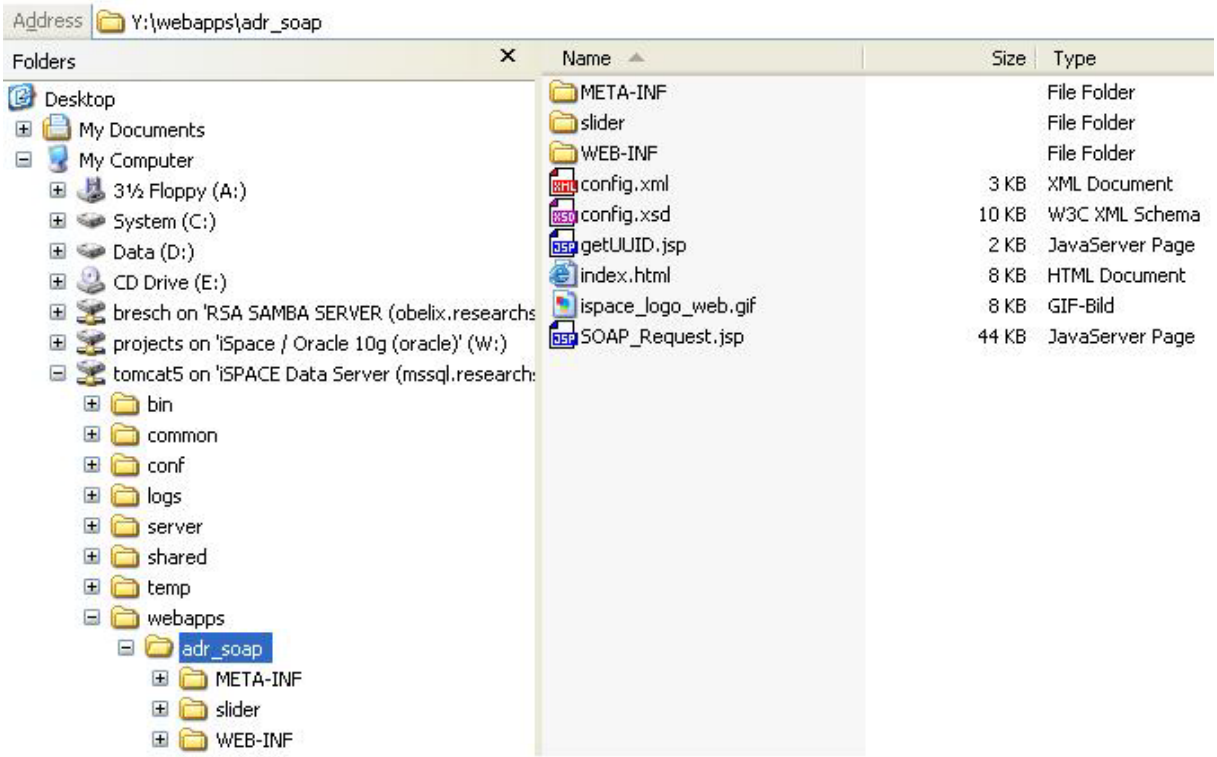

Abbildung 5.2: Verzeichnisstruktur des Tomcat Web-Servers.

Um die Ausführbarkeit der in diesem Kapitel beschriebenen JSP-Pages zu gewährleisten, müssen folgende Bibliotheken importiert werden:

.jar-Datei	Paket	Notwendigkeit
activation.jar	JWSDP (Java Web Service Developer Pack)	javax.activation.DataSource
jaxb-xjc.jar	JWSDP	org.apache.xerces.dom.DocumentImpl
jaxm-api.jar	JAXM	javax.mail.messaging
mail.jar	javamail	javax.mail.internet
saaj-api.jar	JWSDP	javax.api.soap.*
saaj-impl.jar	JWSDP	com.sun.xml.messaging.saaj.client.p2p. HttpSOAPConnectionFactory
JRE System Library	J2SE	java.net.URL , java.util.* + weitere

Tabelle 5.1: Notwendige Bibliotheken für die JSP Seite.

Nach dieser technischen Beschreibung der JSP-Page ist es noch wichtig, die Funktionalitäten und Implementierungsdetails des Servlets zu beleuchten.

5.4 Servlet

Der erste Ausführungsschritt des Servlets ist, wie bei der JSP-Seite, das Auslesen der Elemente der Konfigurationsdatei. Darauf hin nimmt es den SOAP-Request von der JSP-Page entgegen, parst ihn und kreiert mit den erhaltenen Daten eine Datenbankabfrage. Mit den Ergebnisdaten, wird wiederum eine SOAP-Message gebildet, die an die JSP-Seite zurückgesendet wird.

5.4.1 Implementierung

Nach dem Auslesen der Konfigurationsdatei wird der so genannte Servlet-Context, also der Pfad, in dem das Servlet liegt, festgestellt. Dies geschieht mit der Funktion

```
String servlet_context = getServletContext().getRealPath("/");
```

Dieses Feature ist notwendig, um eine Installation des Servlets unabhängig vom Server zu ermöglichen und lediglich das Config-File angepasst werden muss.

Nach dem Festlegen dieser Konfigurationsparameter wird so wie in Kapitel 5.3.1 als Erstes eine leere SOAP-Nachricht erstellt:

```
SOAPFactory receivedSoapFactory = SOAPFactory.newInstance();
MessageFactory receivedFactory = MessageFactory.newInstance();
SOAPHeader receivedHeader = receivedMessage.getSOAPHeader();
SOAPBody receivedBody = receivedMessage.getSOAPBody();
Name receivedBodyName = receivedSoapFactory.createName("getAddress",
 "coords", "http://mssql.researchstudio.at:8081");
```

Als Nächstes werden die Elemente aus dem Header ausgelesen:

```
int session_id = Integer.parseInt(receivedHeader.getAttribute("sid"));
String UUID = receivedHeader.getAttribute("uuid");
```

Danach werden die Kindelemente des Body-Elements herausgefiltert und in einen so genannten Iterator geschrieben, über den die Elemente dann sequentiell mit `iterator.next()` angesprochen werden können.

```
Iterator iterator = receivedBody.getChildElements(receivedBodyName);
SOAPBodyElement bodyElement = (SOAPBodyElement)iterator.next();
```

Dann werden die Attribute in eine `NodeList`-Variable geschrieben. Auf deren Elemente kann dann mittels der Befehle in Zeilen zwei und drei zugegriffen werden.

```
NodeList list = bodyElement.getChildNodes();
list0 = list.item(0).getChildNodes();
value0 = list0.item(0).getNodeValue();
```

An dieser Stelle wird eine Datenbankabfrage zur Benutzerauthentifizierung durchgeführt. Die Werte, die für den Verbindungsaufbau relevant sind (Treiber, Connect-String, Benutzername, Passwort, Spaltennamen), wurden bereits zuvor aus der Konfigurationsdatei ausgelesen. Zur Veranschaulichung ist der relevante Config-File-Ausschnitt in der folgenden Abbildung dargestellt. Die abgebildete Baumdarstellung, wie sie auch in späteren Kapiteln der Diplomarbeit vorkommt, wurde automatisch vom das Programm *XMLSpy* erzeugt. Die einzelnen Elemente werden in Kapitel 7.1 näher behandelt.

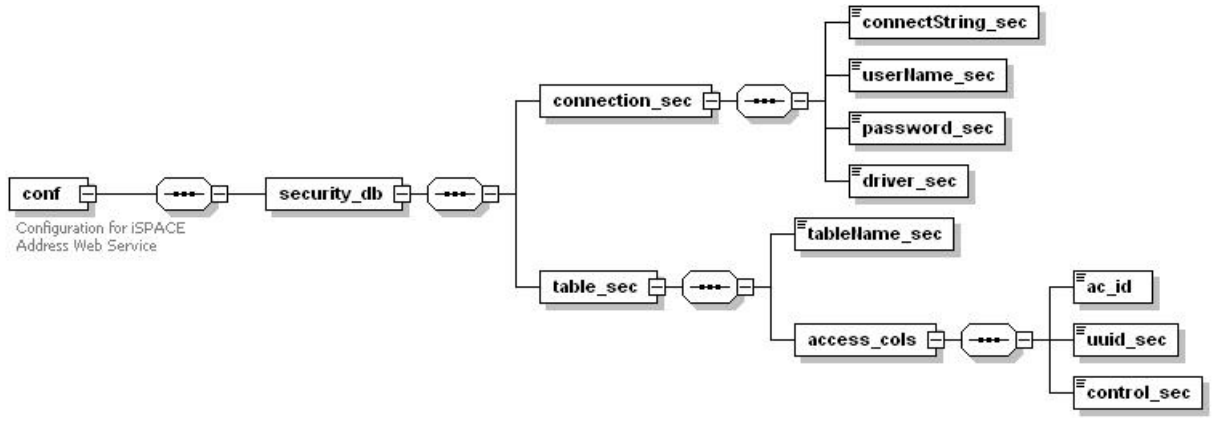

Abbildung 5.3: Zugriffsparameter für die Security-Datenbank.

Die Parameterübergabe an die Funktion `createStatement()` ist notwendig um die etwas gehaltlose SQLException „Ungültiger Vorgang bei Nur-Weiterleiten-Ergebnismenge" zu vermeiden.

```
Class.forName(driver_sec).newInstance();
udb_conn = DriverManager.getConnection(connectString_sec,
 userName_sec, password_sec);
Statement user_stmt = udb_conn.createStatement
 (ResultSet.TYPE_SCROLL_INSENSITIVE, ResultSet.CONCUR_READ_ONLY);
String query_user_db = "select ac." + control_sec + " from " +
 tableName_sec + " ac where ac." + uuid_sec + "='" + UUID + "'";
ResultSet user_rs = user_stmt.executeQuery(query_user_db);
int control = user_rs.getInt(1);
udb_conn.close();
```

Wenn die Benutzerauthentifizierung erfolgreich abgeschlossen ist, wird die eigentliche Datenbankabfrage, die mit JDBC realisiert wird, durchgeführt. Sie ist in den folgenden Abschnitten dargestellt.

Die Verbindungsherstellung funktioniert gleich wie in oben genannter Abfrage. Der Aufbau des *select*-Statements geschieht jedoch dynamisch. Dies bedeutet, dass zurückzugebende Elemente nur dann in die Query eingebaut werden, wenn die entsprechende Spalte auch wirklich existiert, also im Config-File festgelegt ist. Welche Spalten theoretisch abgefragt werden können, ist in der folgenden Abbildung zu sehen:

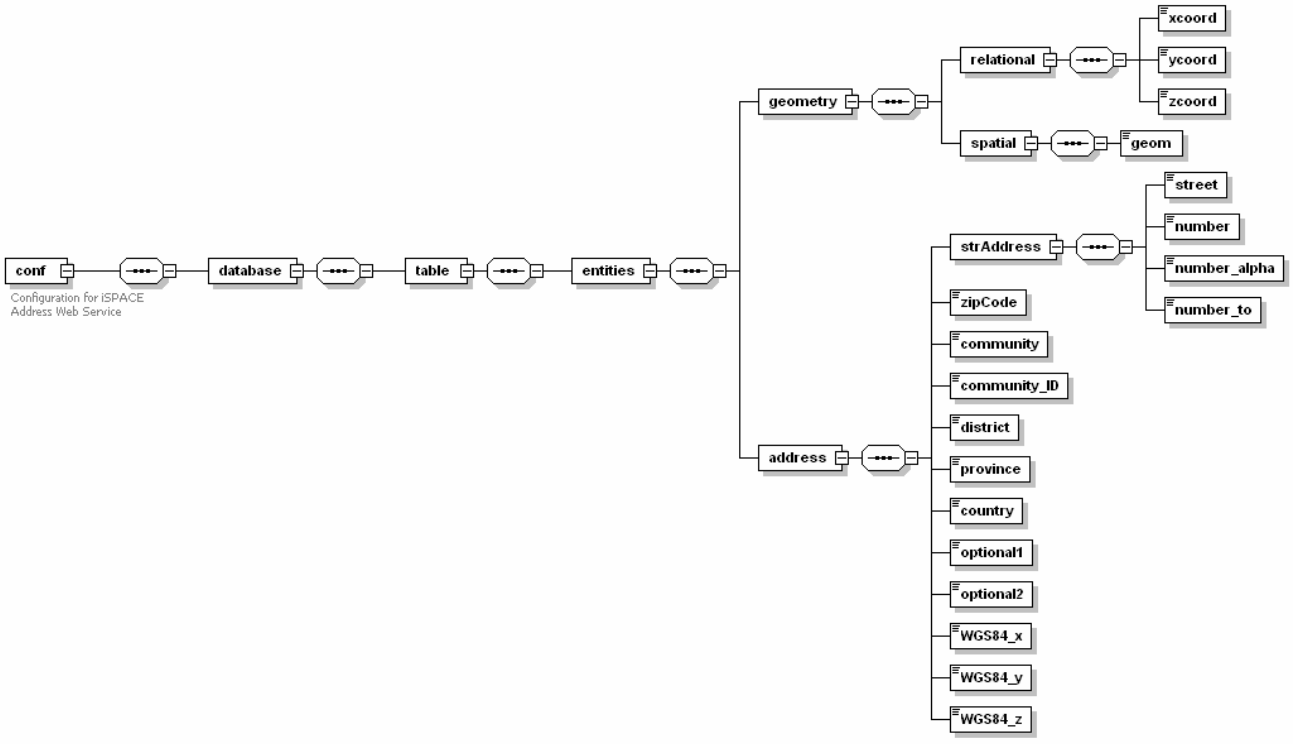

Abbildung 5.4: Config File Auszug der Abzufrageneden Datenbankelemente.

Welche Elemente die Abfrage einschränken, wird durch die Überprüfung, ob die jeweilige Spalte vorhanden ist und zugleich die Variable einen Wert enthält, festgelegt. Eine beispielhafte Adressabfrage könnte folgendermaßen gebildet werden:

```
String query = "select " + "a." + street + "a." + number + "a." +
 number_alpha + " from " + tableName + " a where " + v1 + "=a." +
 xcoord + " and " + v2 + "=a." + ycoord;
```

Die Variable `v1` entspricht dem ersten übergebenen Wert, in diesem Fall der x-Koordinate, `v2` dem zweiten, also der y-Koordinate. Die Abfrage selbst wird dann wieder wie oben durchgeführt. Nach Ende des Datenbankzugriffs wird die Verbindung mittels des folgenden Befehls wieder geschlossen:

```
Statement stmt = conn.createStatement
 (ResultSet.TYPE_SCROLL_INSENSITIVE, ResultSet.CONCUR_READ_ONLY);
ResultSet rs = stmt.exeuteQuery(query);
conn.close();
```

Zusätzlich werden noch in einem *Hashtable* die in die Query eingebauten Elemente samt zugehörigem Index gespeichert, um die eindeutige Element-Position-Zuordnung nicht zu verlieren, die später für die Erstellung der SOAP-Response, die immer die gleiche Form hat, wichtig ist.

Liefert die Datenbankabfrage bei der Adresssuche kein Ergebnis zurück, wird je nach Config-File Eintrag eine Umgebungssuche durchgeführt. Diese kann entweder im Umkreis mit Oracle-Spatial, oder in einem Quadrat rund um den angegebenen Punkt ablaufen.

Verläuft die Koordinatensuche bei eingegebener Adresse erfolglos, wird der Adress-String halbiert und eine zweite Datenbankabfrage abgesetzt.

Eine wichtige Begrenzung, die aus Performance-Gründen gesetzt wurde, ist die maximale Resultatanzahl der zurückgegebenen Reihen, die im Konfigurations-File eingestellt werden kann. Dadurch verläuft die Kommunikation ressourcenschonend und die Ausgabe wird übersichtlich gehalten.

Nach der Datenbankabfrage wird auf Wunsch (Parametersetzung im Config-File) eine Koordinatentransformation über ein extern einzubindendes Web-Service durchgeführt. Die im Eingabeformat abgefragten Koordinaten werden in WGS84-Darstellung transformiert, um ein zusätzliches global verbreitetes, vergleichbares Ausgabeformat, z.B. das Global Positioning System (GPS), zu schaffen.

Nach dieser Prozedur wird das aus der Datenbankabfrage resultierende Ergebnis vom Servlet wieder in eine SOAP-Message eingepackt und an die JSP-Seite geschickt.

5.4.2 Technische Voraussetzungen

Auf dem Web-Server müssen sich im *classes*-Verzeichnis der Webapplikation die Klassendateien befinden, also die Servlets und andere Java-Klassen. Die für die Datenbankverbindung benötigten jar-Dateien müssen ins Verzeichnis *WEB-INF/lib* gelegt werden. Diese Ordnerstruktur wird im .war-File - von *deploytool* generiert - automatisch hergestellt. Sie wird in Kapitel 6.1 noch näher behandelt.

Die in Tabelle 5.1 aufgelisteten Bibliotheken sind allesamt auch in das Servlet-Programm einzubeziehen. Auf Grund der Datenbankabfrage, die für alle gängigen Datenbanken durchführbar ist, und der Notwendigkeit, den SOAP-Request zu parsen, müssen jedoch noch weitere jar-Dateien eingebunden werden:

.jar-Datei	Paket	Notwendigkeit
dom.jar	JWSDP	org.w3c.dom.ranges.DocumentRange
servlet-api.jar	JWSDP	javax.servlet.*'
xercesImpl.jar	JWSDP	org/apache/xerces/dom/DocumentImpl
JRE System Library	J2SE	java.sql.* + einige andere
ojdbc14_g.jar	ojdbc14_g.jar	Oracle Datenbanktreiber
msbase.jar	mssql_JDBC_driver_ setup.exe	MSSQL Datenbanktreiber
msutil.jar		
mssqlserver.jar		
mysql-connector-java-3.0.14-production-bin.jar	mysql-connector-java-3.0.14-production.zip	MySQL Datenbanktreiber
db2cc.jar	db2cc.jar	DB2 (IBM) Datenbanktreiber
db2java.jar	db2java.jar	

Tabelle 5.2: Notwendige Bibliotheken für das Servlet.

5.5 Ausgabe des Ergebnisses

Die JSP-Page parst letzlich die SOAP-Response, wie in Kapitel 5.4.1 dargestellt, und gibt das Resultat aus. Der statische Aufbau der SOAP-Nachricht wird in Kapitel 7 dargestellt.

6 Spezielle Details zur Implementierung

Dieses Kapitel befasst sich mit Besonderheiten in der Implementierung, die während der Entwicklungsphase wichtige Rollen spielten.

6.1 Deploytool

Dieses Programm dient unter anderem dazu, so genannte WAR-Files (Web-Application-Archives) zu erstellen und diese auf einem Web-Server zu installieren.

6.1.1 Struktur eines WAR-Files

WAR-Dateien beinhalten alle für eine Webapplikation nötigen Files. Dies können HTML-, XML-, JSP-, jar-, Servletdateien oder jegliche andere Typen sein. Die Dateistruktur, die auch auf den Web-Server entpackt wird, sieht folgendermaßen aus:

Abbildung 6.1: Ordnerstruktur eines WAR-Files.

Das Wurzelverzeichnis, in diesem Fall *adr_soap*, trägt den Namen des war-Files. Darin gibt es einen Unterordner Namens *WEB-INF*. Dieses Verzeichnis enthält wiederum zwei Ordner. Im *classes*-Verzeichnis befinden sich alle Java-Klassen (Servlets und übrige Klassen). Die JAVA-Dateien selbst müssen nicht in die Webapplikation aufgenommen werden. Das zweite Unterverzeichnis trägt den Namen *lib*. Es enthält alle für das Web- Service nötigen Klassenbibliotheken (jar-Dateien). Alle übrigen Dateien wie JSP-, HTML-, Bilddateien etc. befinden sich direkt im root-Ordner.
Eine Datei, die im Zusammenhang mit Web-Services nicht fehlen darf, ist der so genannte Deployment Descriptor (*WEB-INF/web.xml*). Er enthält eine Beschreibung der Servlets und anderer Komponenten wie JSPs etc. Darin wird festgehalten, wie die Servlet-Klassen bzw. JSP-Files heißen und mit welchem Aliasnamen sie im Web-Service angesprochen werden können. Weiters werden Initialisierungsparameter angegeben und Sicherheitsbedingungen festgelegt, die der Server erfüllen soll.

6.1.2 Erstellen eines WAR-Files

Beim Erstellen einer WAR-Datei muss zuerst eine „Web Application“ kreiert werden. Danach werden die relevanten Dateien in der grafischen Benutzeroberfläche hinzugefügt. Diese werden dann automatisch in die entsprechenden Verzeichnisse gelegt, damit die oben genannte Ordnerstruktur eingehalten wird. Danach können der Applikation „Web Components“ hinzugefügt werden. Bei der Erstellung der war-Datei wird auch der *Deployment Descriptor* automatisch erzeugt, wobei die Servlet-Mappings noch manuell eingetragen werden müssen.
Um ein *Web Application Archive* auf einem Server zu installieren, der auf dem momentan verwendeten Rechner läuft, kann das *deploytool* verwendet werden. Dieses Programm funktioniert jedoch nicht, um das war-File auf einem anderen Server zu entfalten.
Die Vorgangsweise beim in diesem Projekt verwendeten Tomcat Server ist folgende:

- Ändern der Projektdateien
- Update im *deploytool*
- Speichern und damit erzeugen des war-Files
- Entfernen der Webapplikation am Tomcat Server
- Entpacken der war-Datei am Server (simpler Reload funktioniert nicht)

6.2 Besonderheiten von JSP

JSP unterscheidet sich in einigen Punkten von der herkömmlichen Java-Syntax. Einige wichtige Verschiedenheiten werden hier aufgelistet

Erstens müssen Funktionen im so genannten Deklarationstag definiert werden. Dies bedeutet, dass die Methodendefinition am Anfang des JSP-Files innerhalb der folgenden Zeichenfolge stehen muss:

```
<%!
  ...
%>
```

Der zweite zu erwähnende Punkt ist, dass für die Ausgabe (*out.println*) innerhalb einer Funktion die Variable *JspWriter out* der Methode mitübergeben werden muss.

Des Weiteren unterscheiden sich die Importe der Klassenbibliotheken in Java und in JSP:

Java: `import javax.xml.soap.*;`
JSP: `<%@ page import="javax.xml.soap.*" %>`

7 Benutzerinformationen

Dieser Abschnitt beinhaltet eine Beschreibung der für die Benützung des Web-Service relevanten Elemente. Dazu gehören sowohl das Konfigurations-File, die SOAP-Nachrichten selbst und diverse kleinere Aspekte.

7.1 Konfigurationsdatei

Als erstes wird das Config-File beschrieben. Die Struktur der XML-Datei ist in der folgenden Abbildung dargestellt:

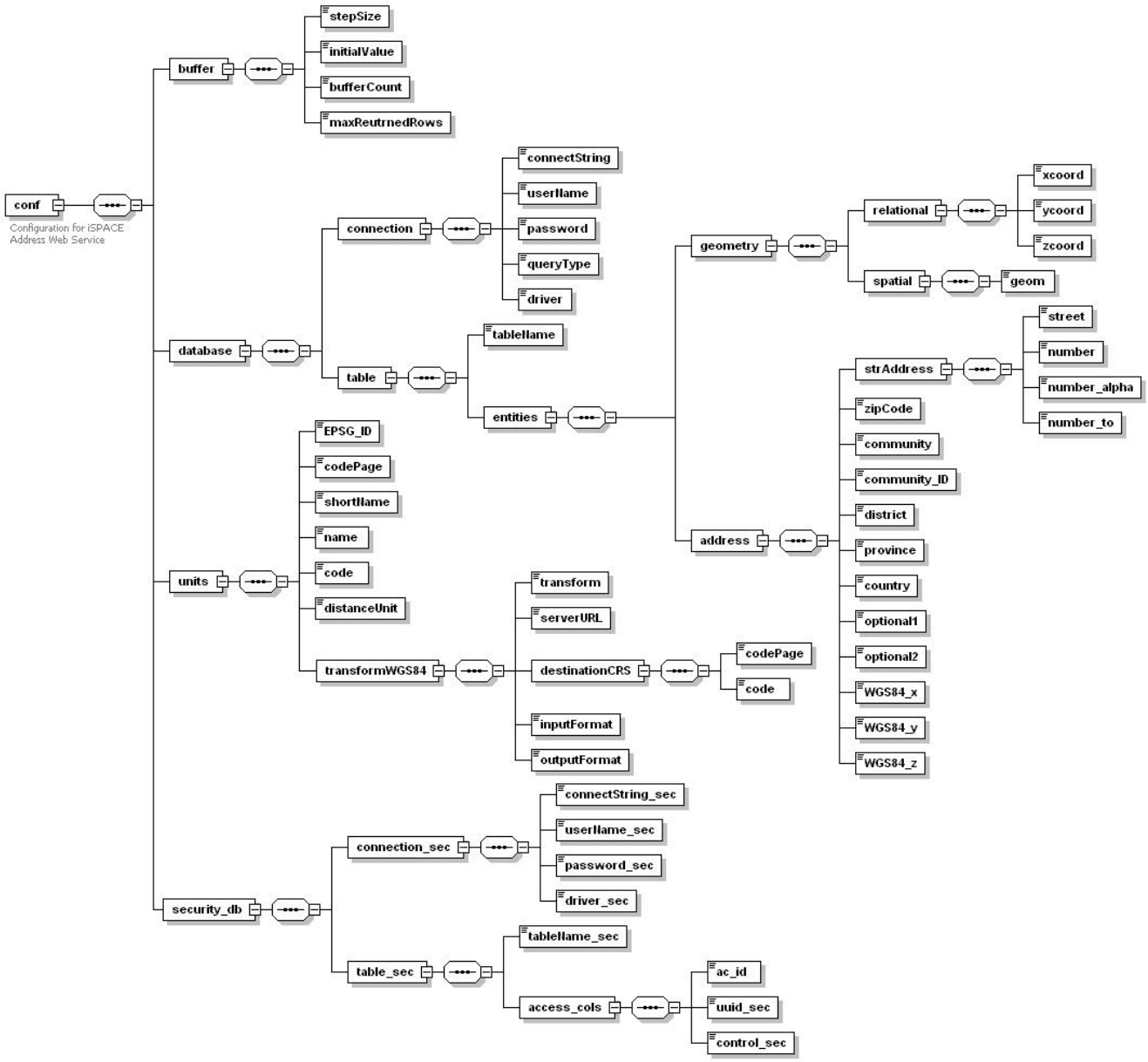

Abbildung 7.1: Struktur des XML-Konfigurationsfiles.

7.1.1 Beschreibung der Elemente

In der folgenden Tabelle werden die Subelemente der vier Grundelemente beschrieben:

Elementname	Beschreibung
buffer	
stepSize	Schrittgröße bei Umkreissuche
initialValue	Startwert der Umkreissuche
bufferCount	Zähler für die Erhöhung der Umkreisgröße
maxReturnedRows	maximale Anzahl der Rückgabedatenreihen
database	
connectString	Connect-String für die DB-Verbindung
userName, password	Benutzername und Passwort für die DB
queryType	Angabe, ob Verwendung von x-, y-, und z-Koordinaten (*relational*) oder Oracle Spatial (*spatial*)
driver	Datenbanktreiber laut Auswahl (Oracle, MySQL, MSSQL, DB2, ODBC)
tableName	Tabellenname für die Abfrage
xcoord, ycoord, zcoord	Spaltennamen für die x-, y- und z-Koordinaten
street, number, number_alpha, number_to	Spaltennamen für die Straße, Hausnummer, alphanumerischen Zusatz der Nummer und „Nummer bis“
zipCode, community, community_ID	Spaltennamen für die Postleitzahl, Gemeinde und Gemeindekennzahl
district, province, country	Spaltennamen für den Bezirk, Region und Land
optional1, optional2	Spaltennamen für optionale Felder
WGS84_x, WGS84_y, WGS84_z	Spaltennamen für x-, y- und z-Koordinaten in WGS84-Form
units	
EPSG_ID	Kennzahl des EPSG-Codes
codePage	Eigentümer des Codes
shortName	Kurzname des Codes
name	vollständiger Name des Codes
code	Coordinate Reference System – Well-Known Text
distanceUnit	Maßeinheit des CRS
transform	Parameter, ob Koordinatentransformation durchgeführt werden soll
serverURL	URL des Servers für die Koordinatentransformation
codePage, code	s. o.
inputFormat, outputFormat	Formate für Koordinatentransformation
security_db	
connectString_sec, userName_sec, password_sec, driver_sec, tableName_sec	Parameter für den Verbindungsaufbau zur Benutzerdatenbank (Beschreibung s.o.)
ac_id	Spaltenname der lfd. Nummer Access ID
uuid_sec	Spaltenname der UUID des Benutzers
control_sec	Spaltenname der Berechtigungsvariable

Tabelle 7.1: Elemente des Config-Files.

7.2 Struktureller Aufbau der SOAP-Nachrichten

Nach der Erstellung des Konfigurations-Files kann das Web-Service mit Hilfe von SOAP angesprochen werden. Sowohl die Anfrage (je nach Abfragetyp, s.u.) als auch die Antwort haben eine vorgegebene Struktur, was eine konsistente und leicht nutzbare Schnittstelle nach außen hin gewährleistet. Die Erzeugung und das Parsen der SOAP- Messages mit Java geschieht analog zu den in Kapitel 5.3.1 und 5.4.1 beschriebenen Codeabschnitten.

7.2.1 SOAP-Request

Die Anfrage muss neben dem standardisierten SOAP-Envelope folgende Elemente und Attribute enthalten, um vom Web-Service bearbeitet werden zu können.
Erstens muss der SOAP-Header zwei Attribute enthalten: erstens *uuid*, das im Servlet zur Benutzerdatenbankabfrage herangezogen wird und außerdem eine Session-ID *sid* zur Identifizierung der Abfrage.
Der Body enthält ein Element, in dem alle anderen Elemente verpackt sind. An dieser Stelle wird bei der Erzeugung der Nachricht unterschieden, ob nach einer Adresse oder nach Koordinaten gesucht wird.
Im Fall der Adresssuche ist das erste Unterelement immer der Variablenwert für die Umkreissuche. Darauf folgt ein Element, in dem eingegebenen Koordinaten GML-konform verpackt sind.
Bei der Suche nach Koordinaten werden anstatt des GML-Elements die fünf Elemente *street, number, number_alpha, community* und *zipCode* mitgegeben. Eine Beschreibung dieser Elemente kann in Kapitel 7.1.1 gefunden werden. Wenn die jeweilige Variable keinen Wert enthält, wird ein Leerzeichen mitgeschickt. Falls ein Element, also die Spalte in der Datenbank nicht vorhanden ist, wird es als „*empty“* notiert.
Die SOAP-Requests für Adress- bzw. Koordinatenabfrage könnten also beispielsweise so aussehen:

Adressabfrage:

```
<SOAP-ENV:Envelope xmlns:SOAP-ENV=
"http://schemas.xmlsoap.org/soap/envelope/">
    <SOAP-ENV:Header sid="-1632754207" uuid="f8fe9120-dbeb-11d8-9669-
    0800200c9a66"/>
       <SOAP-ENV:Body>
          <coords:getAddress
          xmlns:coords="://mssql.researchstudio.at:8081">
             <gml:location xmlns:gml="http://www.opengis.net/gml">
                <gml:Point srsName="EPSG:31289">
```

```
                    <gml:pos>-22049 292997 </gml:pos>
                </gml:Point>
            </gml:location>
        </coords:getAddress>
    </SOAP-ENV:Body>
</SOAP-ENV:Envelope>
```

Aus obiger SOAP-Struktur ist ersichtlich, dass der Punkt, also die x-, y- und z-Koordinaten standardisiert in den *gml:location*-Tag eingebaut sind. Die einzelnen Komponenten sind durch Leerzeichen voneinander getrennt. Im hier gezeigten Beispiel ist kein Wert für die z-Koordinate vorhanden.

Koordinatenabfrage:

```
<SOAP-ENV:Envelope xmlns:SOAP-ENV=
"http://schemas.xmlsoap.org/soap/envelope/">
    <SOAP-ENV:Header sid="-1632754207" uuid="f8fe9120-dbeb-11d8-9669-
    0800200c9a66"/>
    <SOAP-ENV:Body>
       <coords:getAddress
       xmlns:coords="://mssql.researchstudio.at:8081">
          <STR_NAME>ainringweg</STR_NAME>
          <HNR>13</HNR>
          <HNR_ALPHA> </HNR_ALPHA>
          <empty> </empty>
          <empty> </empty>
       </coords:getAddress>
    </SOAP-ENV:Body>
</SOAP-ENV:Envelope>
```

Diese Beispielstruktur bedeutet, dass nach der Adresse „Ainringweg 13“ gesucht wird. Die Spalte für den alphanumerischen Zusatz der Hausnummer ist zwar vorhanden, es wurde aber kein Wert dafür eingegeben. Die folgenden zwei *empty*-Elemente bedeuten, dass die Spalten für den Ort und die Postleitzahl in der Datenbank laut Config-File nicht existieren.

7.2.2 SOAP-Response

Im Gegensatz zur Anfrage hat die Antwort bei beiden Abfragetypen eine fixe Struktur. Die ersten beiden Elemente im *coords*-Element sind bei der Response die Anzahl der Reihen des Ergebnisses und der aktuelle Wert der Umkreissuche. Die Struktur der Antwort sieht folgendermaßen aus:

```
<SOAP-ENV:Envelope xmlns:SOAP-
ENV="http://schemas.xmlsoap.org/soap/envelope/">
   <SOAP-ENV:Header compass="false" control="1" sid="429714959"/>
```

```
<SOAP-ENV:Body>
   <coords:getAddressResponse
   xmlns:coords="://mssql.researchstudio.at:8081">
      <rowCount>15</rowCount>
      <buffer_val>50</buffer_val>
      <gml:location xmlns:gml="http://www.opengis.net/gml">
         <gml:Point srsName="EPSG:31289">
            <gml:pos>-22049 292997  </gml:pos>
         </gml:Point>
      </gml:location>
      <STR_NAME>Ainringweg</STR_NAME>
      <HNR>13</HNR>
      <HNR_ALPHA>B</HNR_ALPHA>
      <empty> </empty>
      <ZIP>5020</ZIP>
      <CITY>Salzburg</CITY>
      <empty> </empty>
      <BEZIRK>Salzburg</BEZIRK>
      <empty> </empty>
      <COUNTRY>Österreich</COUNTRY>
      <empty> </empty>
      <empty> </empty>
      <gml:location xmlns:gml="http://www.opengis.net/gml">
         <gml:Point srsName="EPSG:4326">
            <gml:pos>16.0391074540754 47.775758835661  </gml:pos>
         </gml:Point>
      </gml:location>
   </coords:getAddressResponse>
</SOAP-ENV:Body>
</SOAP-ENV:Envelope>
```

Die Struktur der SOAP-Response entspricht einem Teil der Elementliste (Liste der Spaltennamen), die in Kapitel 7.1.1 aufgeführt ist. Beim Aufbau der Antwort werden wiederum gegebenenfalls Elemente, die nicht in der abgefragten Tabelle vorhanden sind, als „*empty*“ eingeführt.

Näherer Erklärung bedarf noch das letzte Element im *coords*-Tag. Dieses zweite GML-Element enthält transformierte WGS84-Koordinaten, die von einem standardisierten Koordinatentransformationsservice (Open GIS Consortium, Web Coordinate Transformation Service Version 1.0) erzeugt werden.

7.3 Weitere Informationen

Einige weniger umfangreiche aber für die Benützung des Web-Service dennoch sehr wichtige Informationen sind in diesem Unterkapitel beschrieben.

7.3.1 Verteilte Suche

Das Web-Service bietet die Möglichkeit, eine Abfrage in mehreren Datenbanken abzusetzen. Dazu muss das Servlet, inklusive des dazugehörigen Konfigurations-Files, lediglich auf verschiedenen Web-Servern installiert und angesprochen werden. Hierzu sind keine Änderungen im Code nötig.
In Verbindung mit der aktuellen Version der JSP-Seite können bis zu zwei verteilte Datenbanken gleichzeitig angesprochen werden. Zur Benützung dieses Features muss im JSP-Code die URL des Web-Servers eingegeben werden, auf dem das Servlet installiert wird, von dem aus dann die alternative Datenbank angesprochen wird. Um diese Funktionalität zu erweitern sind nur einige kleine Veränderungen im JSP-Code notwendig.

7.3.2 Unterstützung Aller Gängigen Datenbanktypen

Wie bereits erwähnt, können alle gängigen Datenbanken in Verbindung mit dem Web-Service verwendet werden. Die entsprechenden Daten für die Verbindung müssen lediglich in der Konfigurationsdatei eingegeben werden. In diesem stehen folgende Datenbanktypen bzw. -mechanismen zur Auswahl: Oracle, MSSQL Server, MySQL, DB2 und ODBC.

8 Schlussfolgerung

Das in dieser Diplomarbeit beschriebene Web-Service für den standardisierten Austausch von Koordinaten- und Adressdaten beinhaltet eine Reihe von Fähigkeiten, die in den Kapiteln 4 und 5 detailliert beschrieben sind.

Prinzipiell bietet das Service eine automatisierte Abbildung einer Adresse zu geografischen Koordinaten. Auf Grund des in Kapitel 1 beschriebenen Hintergrundes des Einsatzes in Katastrophenszenarien erfolgte die Realisierung des Service mittels SOAP als Kommunikationsprotokoll, JSP für die Verwendung von dynamischen Elementen in HTML-Seiten und Servlets, um individuelle Benutzeranfragen dynamsisch am Server bearbeiten zu können. Das Web-Service selbst wurde unter Verwendung des Java Web Services Developer Pack kreiert.

Mit Hilfe diverser Erweiterungen wurde die praktische Anwendbarkeit des Service gesteigert. Solche zusätzliche Features sind z.B. diverse Toleranzmechanismen bei der Adresseingabe, eine Umkreissuche bei ungenauer Koordinateneingabe, eine XML-Konfigurationsdatei, die eine Nutzung des Service ermöglicht, ohne die enthaltenen Programme selbst verändern zu müssen, eine standardisierte, GML-konforme Koordinatenrückgabe, eine UUID zur Benutzer-authentifizierung und die Nutzung eines Koordinatentransformationsservice, um Positionen in der vom Benutzer gewünschten Form ausgeben zu können.

Mögliche Erweiterungen des im *adr_soap*-Projekt erstellen Web-Service werden im folgenden Kapitel erläutert.

9 Ausblick

Abschließend werden einige Aspekte erläutert, die als zusätzliche Weiterentwicklungen das *adr_soap*-Projekt komplettieren und es damit dem Praxiseinsatz einen Schritt näher bringen würden.

9.1 WSDL und UDDI

Durch WSDL speziell in Verbindung mit UDDI wäre eine Möglichkeit gegeben, das Web-Service öffentlich verfügbar zu machen und die in diesem Bericht beschriebenen Kriterien zum Ansprechen der Applikation standardisiert potenziellen Benutzern zur Verfügung zu stellen. Dies würde eine hervorragende Grundlage für eine weit verbreitete Verwendung des Service darstellen.

9.2 Ergänzungen zum Koordinatentransformationsservice

In der jetzigen Variante wird ein externes Koordinatentransformationsservice angesprochen. Um die Response-Zeiten bei der Adressabfrage möglichst gering zu halten, wird die Integration von eigenen Klassenbibliotheken (jar) in einer Weiterentwicklung der Anwendung angestrebt, wie sie beispielsweise in [GEOT04] zur Verfügung gestellt werden.

Literaturverzeichnis

[ALTO04] Altova GmbH (2004) ALTOVA - XML Development, Data Mapping, and Content Authoring Tools. http://www.altova.com (2. Juni 2004).

[APAC04] The Apache Software Foundation (2004) The Jakarta Site - Apache Tomcat. http://jakarta.apache.org/tomcat/index.html (2. Juni 2004).

[ASSI04] Assisi, R. (2004) *Eclipse – Einführung und Referenz – Java-Entwicklung mit der Open-Source-Plattform*. Carl Hanser Verlag, ISBN 3-446-22620-6, München, Wien, 2004.

[AYER99] Ayers, D. et al. (1999) *Professional Java Server Programming*. Wrox Press Ltd, ISBN 1-861002-77-7, Birmingham, Großbritannien, 1999.

[BIRO04] Biron, P. und Malhotra, A. (Editoren) (2004) XML Schema Part 2: Datatypes Second Edition. http://www.w3.org/TR/xmlschema-2, W3C Recommendation Paper, Zweite Ausgabe, 28. Oktober 2004 (16. Februar 2005).

[BODO04] Bodoff, S. (2004) Java Servlet Technology. http://java.sun.com/webservices/docs/1.0/tutorial/doc/Servlets.html (2. Juni 2004).

[BOX00] Box, D. et al. (2000) Simple Object Access Protocol (SOAP) 1.1. http://www.w3.org/TR/2000/NOTE-SOAP-20000508, W3C Note, 8. Mai 2000 (2. Juni 2004).

[BRAY04] Bray, T. et al. (2004) Extensible Markup Language (XML) 1.0 (Third Edition). http://www.w3.org/TR/REC-xml, W3C Recommendation, Dritte Ausgabe, 4. Februar 2004 (16. Februar 2005).

[CHRI01] Christensen, E. et al. (2001) Web Services Description Language (WSDL). http://www.w3.org/TR/wsdl, W3C Note, 15. März 2001 (18. August 2004).

[COX04] Open GIS Consortium (2004) *OpenGIS® Geography Markup Language (GML) - Implementation Specification*. OpenGIS Recommendation Paper, ISO/TC 211/WG 4/PT 19136, 7. Februar, 2004.

[ECLI04] Eclipse Foundation (2004) Eclipse.org Main Page. http://www.eclipse.org (2. Juni 2004).

[FISH04] Fisher, M. (2004) JDBC(TM) Database Access. http://java.sun.com/docs/books/tutorial/jdbc/ (2. Juni 2004).

[GEOT04] GeoTools Project (2004) GeoTools – Home. http://www.geotools.org (18. August 2004).

[GUDG03a] Gudgin, M. et al. (2003) SOAP Version 1.2 Part 1: Messaging Framework. http://www.w3.org/TR/2003/REC-soap12-part1-20030624, W3C Recommendation, 24. Juni 2003, (2. Februar 2005).

[GUDG03b] Gudgin, M. et al. (2003) SOAP Version 1.2 Part 2: Adjuncts. http://www.w3.org/TR/2003/REC-soap12-part2-20030624, W3C Recommendation, 24. Juni 2003 (2. Februar 2005).

[HARO99] Harold, E. R. (1999) *XML Bible*. IDG Books Worldwide, Inc., ISBN 0-7645-3236-7, Foster City, CA 94404, 1999.

[HARO05] Harold, E. R. und Means, W. S. (2005) *XML in a Nutshell*. 3. Auflage, O'Reilly Verlag, ISBN 3-89721-339-7, 2005.

[HEIN03] Hein, M. und Zeller, H. (2003) *Java Web Services – Entwicklung plattformübergreifender Dienste mit J2EE, XML und SOAP*. Addison-Wesley Verlag, ISBN 3-8273-2071-2, München, Deutschland, 2003.

[MINT03] Mintert, S. (2003) *XHTML, CSS und Co. Die W3C-Spezifikationen für das Web-Publishing*. Stefan Mintert, Reihe Edition W3C.de, Addison-Wesley Verlag, ISBN 3827318726, 2003.

[MOCZ05] Moczar, L. (2005) *Tomcat 5 - Einsatz in Unternehmensanwendungen mit JSP und Servlets*. Addison-Wesley Verlag, ISBN 3-8273-2202-2, München, Deutschland, 2005.

[MUEN01] Münz, S. (2001) SELFHTML: XML/DTDs. http://de.selfhtml.org (27. Juli 2004).

[OGBU00] Ogbuji, U. (2000) Using WSDL in SOAP Applications - An Introduction to WSDL for SOAP Programmers. http://www-106.ibm.com, 1. November 2000 (7. Juni 2004).

[REFS05a] Refsnes Data (2004) WSDL Tutorial. http://www.w3schools.com/wsdl (18. August 2004).

[REFS05b] Refsnes Data (2004) SOAP Tutorial. http://www.w3schools.com/soap (18. Jänner 2005).

[STLA98] St. Laurent, S. (1998) *XML: A Primer*. M&T Books, Inc., 352 Seiten, IDG Books Worldwide, ISBN 155828592X, January 1, 1998.

[SUN05a] Sun Microsystems, Inc. (2005) Java Technology. http://java.sun.com (2. Juni 2004).

[SUN05b] Sun Microsystems, Inc. (2005) The Java(TM) Web Services Tutorial. http://java.sun.com/webservices/docs/1.3/tutorial/doc/index.html (2. Juni 2004).

[SUN05c] Sun Microsystems, Inc. (2005) Java Web Services Developer Pack. http://java.sun.com/webservices/jwsdp/index.jsp (30. Jänner 2005).

[THOM04] Thompson, H. S. et al. (2004) XML Schema Part 1: Structures Second Edition. http://www.w3.org/TR/xmlschema-1, W3C Recommendation, Zweite Ausgabe, 28. Oktober 2004 (2. Februar 2005).

[UDDI00] UDDI.ORG Consortium. (2000) UDDI Technical White Paper. http://www.uddi.org/pubs/Iru_UDDI_Technical_White_Paper.pdf, 6. September 2000 (16. Februar 2005).

[WINE98] Winer, D. (1998) RPC over HTTP via XML. davenet.userland.com/1998/02/27/rpcOverHttpViaXml, Posting, 27. Februar 1998, (8. Februar 2005)

Abkürzungsverzeichnis

API	…	Application Programming Interface
AXIS	…	Apache Extensible Interaction System
CGI	…	Common Gateway Interface
CORBA	…	Common Object Request Broker Architecture
CRS	…	Coordinate Reference System
CSS	…	Cascading Style Sheet
DTD	…	Document Type Definition
EPSG	…	European Petroleum Survey Group
GML	…	Geography Markup Language
GPS	…	Global Positioning System
HTML	…	Hypertext Markup Language
HTTP	…	Hyper Text Transfer Protocol
JAF	…	Java Activation Framework
JAXB	…	Java Architecture for XML Binding
JAXM	…	Java API for XML Messaging
JAXP	…	Java API for XML Processing
JAXR	…	Java API for XML Registry
JAX-RPC	…	Java API for XML-based RPC
JDBC	…	Java DataBase Connectivity
JDK	…	Java Development Kit
JSP	…	Java Server Pages
JWSDP	…	Java Web Services Developer Pack
JWSTK	…	Java Web Service Tool Kit
MEP	…	Message Exchange Pattern
PDA	…	Personal Digital Assistant
RPC	…	Remote Procedure Call
SAAJ	…	SOAP with Attachments API for Java
SDK	…	Software Development Kit
SMTP	…	Simple Mail Transfer Protocol
SOAP	…	früher: Simple Object Access Protocol
STL	…	Standard Tag Library
UDDI	…	Universal Description, Discovery and Integration

UML	…	Unified Markup Language
URI	…	Uniform Resource Identifier
URL	…	Uniform Resource Locator
UUID	…	Universal Unique Identifier
W3C	…	World Wide Web Consortium
WAR	…	Web Application Archive
WSDD	…	Web Service Deployment Descriptor
WSDL	…	Web Service Description Language
XLL	…	Extensible Linking Language
XML	…	Extensible Markup Language
XSD	…	Extensible Schema Definition
XSL	…	Extensible Style Language
XSLT	…	XSL-Transformationen