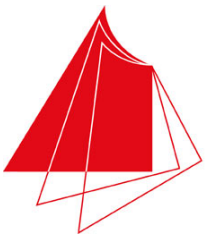

Hochschule Karlsruhe
Technik und Wirtschaft
UNIVERSITY OF APPLIED SCIENCES

Diplomarbeit

im Studiengang Kartographie und Geomatik
Fachbereich Geoinformationswesen

Erstellung eines Konzepts zum Aufbau einer Service-orientierten GIS-Bibliothek für raumbezogene Auswertungen in einem Umweltinformationssystem

Stefan Haberer
Matr.-Nr.: 012570
Wintersemester 2005/2006

Betreuer:
Prof. Dr.-Ing. G. Schweinfurth

LUBW

erstellt an der Landesanstalt für Umwelt,
Messungen und Naturschutz Baden-Württemberg

HOCHSCHULE KARLSRUHE – TECHNIK UND WIRTSCHAFT
Fakultät für Geoinformationswesen – Studiengang Kartographie und Geomatik

Diplomarbeit

für Herrn Stefan H a b e r e r

Thema: Erstellung eines Konzepts zum Aufbau einer Service-orientierten GIS-Bibliothek für raumbezogene Auswertungen in einem Umweltinformationssystem

Die Landesanstalt für Umweltschutz Baden-Württemberg betreibt seit mehreren Jahren ein Umweltinformationssystem (UIS), mit dem neben den Landesdienststellen auch die Stadt- und Landkreise arbeiten. Innerhalb des UIS erfolgt die Erfassung und Auswertung der Geodaten mit Hilfe des Räumlichen Informations- und Planungssystems (RIPS). Das RIPS besteht aus zentral und dezentral eingesetzten Fachanwendungen, in denen derzeit mehrere plattformabhängige GIS-Werkzeuge zum Einsatz kommen. Diese Werkzeuge, v.a. die MS-basierten Produkte ArcWaWiBo, RIPS-Viewer und das JAVA-basierte GISterm, enthalten teilweise identische Funktionen, die jeweils anwendungsabhängig und damit redundant implementiert sind. Da dies mit zahlreichen Nachteilen verbunden ist, besteht das langfristige Ziel, ein auf der .NET-Technologie basierendes RIPS-Framework aufzubauen, um gemeinsam genutzte Geo-Funktionen mit ein- und demselben Werkzeug dem Anwenderkreis zur Verfügung zu stellen. Dabei sollen getrennt voneinander ein Desktop- und ein Webservice-Framework entwickelt werden.

Der Schwerpunkt der Diplomarbeit liegt auf der Konzeption des Webservice-Frameworks. Im ersten Teil soll anhand zweier Testfunktionen ein Weg aufgezeigt werden, wie Webservices auf Basis des ESRI-Produkts ArcGIS Server 9.0 entwickelt werden können. Im Anschluss daran werden verschiedene Test-Klienten erstellt, um zu untersuchen, wie diese Services in .NET-Webapplikationen, HTML/ Javascript- und MS-Office-Anwendungen eingebunden werden können.

Der zweite Teil der Arbeit konzentriert sich auf die prototypische Entwicklung ausgewählter Funktionen des Frameworks. Es soll zunächst untersucht werden, worin der grundsätzliche Nutzen einer GIS-Bibliothek besteht und welche Vor- bzw. Nachteile sich aus deren Einsatz ergeben. Danach sollen die im ersten Teil entwickelten Webservices in die bereits bestehenden Komponenten des Frameworks eingepflegt werden. Darüber hinaus sollen Möglichkeiten aufgezeigt werden, wie die Web Services innerhalb des Frameworks kategorisiert, verwaltet und in einem weiteren Schritt veröffentlicht werden können.

Die Diplomarbeit erhebt nicht den Anspruch, eine performante und robuste Produktionsversion des Frameworks zu entwickeln, zumal noch keine abschließende Bedarfsanalyse der zu integrierenden Funktionalitäten erstellt wurde. Vielmehr soll ein erster Vorschlag für den grundsätzlichen Aufbau einer Basis-Architektur des GIS-Frameworks sowie für ein Vorgehensmodell zur Einbindung und effizienten Pflege weiterer Funktionskomponenten erarbeitet werden.

Die Aufgabe wird am Informationstechnischen Zentrum (ITZ) der LfU Baden-Württemberg mit Sitz in Karlsruhe ausgeführt, ein entsprechender Arbeitsplatz steht bereit.

Bearbeitungszeit: 4 Monate

Ausgabedatum: .....
Abgabedatum: .....

…….....
(Prof. Dr.-Ing. G. Schweinfurth)

Erklärung

Hiermit versichere ich, dass ich diese Diplomarbeit selbstständig verfasst und keine anderen als die von mir angegebenen Quellen und Hilfsmittel benutzt habe.

Karlsruhe, 24. Februar 2006

Stefan Haberer

Danksagung

An dieser Stelle möchte ich mich bei Herrn Prof. Dr.-Ing. Gerhard Schweinfurth für die Betreuung dieser Diplomarbeit bedanken.

Des Weiteren möchte ich mich beim Sachgebietsleiter „Raumbezogene Informationssysteme“, Herrn Biologiedirektor Manfred Müller, bei Herrn Wolfgang Schillinger sowie bei allen Mitarbeitern des Referats 53.2 der Landesanstalt für Umwelt, Messungen und Naturschutz für die fachliche Unterstützung bedanken.

Bedanken möchte ich mich bei Herrn Dieter Kaltenbach von der Firma AHK in Freiburg, der sich stets viel Zeit genommen und mich bei der Durchführung dieser Arbeit unterstützt hat.

Besonderer Dank gilt meiner Familie, die mir das Studium ermöglicht und mich in allen Lagen unterstützt hat.

Inhaltsverzeichnis

1 Einleitung

1.1 Vorwort

Das Internet befindet sich in den letzten Jahren in einem stetigen Prozess der Veränderung. Standen vor geraumer Zeit noch statische und vollkommen autarke Systeme im Vordergrund, so wird die Entwicklung langfristig zu einer immer stärkeren Vernetzung von Anwendungen und Geschäftsprozessen führen. Zukünftige Systeme werden nicht mehr isoliert entwickelt werden, sondern aus verschiedenen Daten und Diensten zusammengesetzt, die über das gesamte Internet verteilt sein können. Die Idee solcher Service-orientierten Architekturen (SOA) ist nicht neu, sondern wurde schon in den 90er Jahren beschrieben und seither immer weiter verfeinert. Als problematisch bei der Einbindung externer Dienste erwies sich in der Vergangenheit jedoch immer, dass sämtliche Applikationen ihre eigenen Schnittstellen verwendet haben. Eine mögliche Antwort auf diese Fragen bieten die sich derzeit in aller Munde befindlichen XML Web Services. Zwar existiert bislang aufgrund fehlender formaler Standards keine endgültige Definition dessen, was sie letztlich alles werden leisten können, dennoch sind sich viele Experten einig, dass diese Technologie die Computerwelt revolutionieren wird. Dabei stellen XML Web Services keine neuartige, revolutionäre Technik dar. Viele der Basistechnologien wie die grundlegenden Internetprotokolle, die verteilte Systemarchitektur und die komponentenbasierte Art der Entwicklung existieren schon seit Jahrzehnten und wurden in offener Zusammenarbeit unterschiedlicher Interessensgruppen entworfen (Freeman & Jones, 2003). Neben der sich daraus ergebenden breiten Akzeptanz bieten Web Services den großen Vorteil, dass sie durch ihre plattformübergreifende Konzeption unabhängig von der Programmiersprache, in der sie erstellt wurden, in die verschiedensten Systeme eingebunden werden können.

1.2 Aufgabenstellung und Zielsetzung

Die Landesanstalt für Umweltschutz Baden-Württemberg betreibt seit mehreren Jahren ein Umweltinformationssystem (UIS), mit dem neben den Landesdienststellen auch die Stadt- und Landkreise arbeiten. Innerhalb des UIS erfolgt die Erfassung und Auswertung der Geodaten mit Hilfe des Räumlichen Informations- und Planungssystems (RIPS). Das RIPS besteht aus zentral und dezentral eingesetzten Fachanwendungen, in denen derzeit mehrere plattformabhängige GIS-Werkzeuge zum Einsatz kommen. Diese Werkzeuge, v.a. die MS-basierten Produkte ArcWaWiBo, RIPS-Viewer und das Java-basierte GISterm, enthalten teilweise identische Funktionen, die jeweils anwendungsabhängig und damit redundant implementiert sind. Da dies mit zahlreichen Nachteilen verbunden ist, besteht das langfristige Ziel, ein auf der .NET-Technologie basierendes RIPS-Framework aufzubauen, um gemeinsam genutzte Geo-Funktionen mit ein- und demselben Werkzeug dem Anwenderkreis zur Verfügung zu stellen. Dabei sollen getrennt voneinander ein Desktop- und ein Webservice-Framework entwickelt werden.

Die Untersuchung wird in verschiedene, aufeinander aufbauende Teilziele untergliedert. Nach einer kurzen Einführung in das UIS, das RIPS und die verschiedenen GIS-Werkzeuge werden mit dem in ArcMap integrierten Visual Basic for Applications (VBA)-Editor zwei Testfunktionen erstellt. Da in VBA ein Zugang zum ArcObjects-Objektmodell existiert, kann ArcGIS auf einfache Weise um Zusatzapplikationen erweitert werden. Die Programmierung in der unmittelbaren Anwendungsumgebung hat den Vorteil, dass die Funktionalität ohne Kompilierung getestet werden kann, zumal die ArcGIS Developer Help bislang nur bedingt Unterstützung für die Entwicklung neuer .NET-Extensions bietet. Diese Phase dient der Einarbeitung in die umfangreiche ArcObjects-Bibliothek und deren Klassendiagramme, wird aber im Rahmen dieser Arbeit nicht weiter erläutert. Der nächste Schritt ist die Portierung der VBA-Funktionen nach .NET. Hierbei soll aufgezeigt werden, was bei der Programmierung von ArcMap-Erweiterungen mit Microsoft Visual Studio und C# zu beachten ist, und wie diese Tools genutzt werden können. Anschließend sollen diese Funktionen als Web Services auf Basis von ArcGIS Server 9.0 umgesetzt werden. Mit der Erstellung verschiedener Test-Clients soll geprüft werden, wie die Web Services in

.NET WebForm-, MS-Office- und HTML/JavaScript-Clients eingebunden werden können.

Der zweite Teil der Arbeit beschäftigt sich mit dem Aufbau des RipsFramework. Zunächst werden die unterschiedlichen Wiederverwendbarkeitstechniken in der objektorientierten Programmierung vorgestellt. Es wird diskutiert, worin die grundsätzlichen Vor- bzw. Nachteile einer GIS-Bibliothek liegen. Im Anschluss soll ein Schubladensystem entworfen werden, um bestehende und zukünftige Webdienste verwalten zu können. Darüber hinaus werden unterschiedliche Standards geprüft, mit denen die Web Services den Anwendern zur Verfügung gestellt werden können. Ein weiterer Aspekt wird die effiziente Fortführung des RipsFramework unter dem Einsatz von Versionskontrollsystemen sein. Abschließend werden verschiedene Maßnahmen vorgestellt, mit denen die Leistungsfähigkeit des GIS Servers und der Web Service-Klassen gesteigert werden kann.

1.3 Das Umweltinformationssystem Baden-Württemberg (UIS)

Heutzutage besteht verstärkt die Anforderung, umweltrelevante Daten nicht isoliert, sondern im Zusammenhang zu betrachten. Im Umweltinformationssystem des Landes Baden-Württemberg wird die gesamte Verarbeitung von Umweltinformationen unter der Federführung des Ministeriums für Umwelt und Verkehr ressortübergreifend koordiniert. Daten zu verschiedenen Themen wie Wasser, Luft, Abfall, Natur oder Landschaft werden von den jeweiligen Stellen erfasst und gepflegt. Der Einsatz moderner Computernetzwerke und einheitlicher Standards stellt dabei sicher, dass sämtliche Daten themenübergreifend verwendet werden können und somit eine ganzheitliche Betrachtung der Problemstellungen ermöglichen. Die Aufgaben des UIS lassen sich in folgende Teilgebiete untergliedern:

1. Planung und Verwaltungsvollzug

   ⇒ Einsatz der Informationstechnik zur effektiven Erledigung der Verwaltungsaufgaben

2. Umweltbeobachtung, Monitoring

   ⇒ Ermittlung, Analyse und Prognose der punktuellen und landesweiten Umweltsituation

3. Integration und Investitionsschutz

   ⇒ Koordination und Integration der vorhandenen Verfahren zur Umweltinformation

4. Notfallmanagement

   ⇒ Bewältigung von Not-, Stör- und Vorsorgefällen durch Nachrichtenvermittlung und –verarbeitung

5. Information, Berichterstattung

   ⇒ Information der politischen Führung, der Verwaltung, des Bundes und der Europäischen Union sowie der Öffentlichkeit

An der Entwicklung des UIS sind neben dem Umweltministerium auch alle weiteren Landesministerien bzw. deren Fachbehörden für Umweltaufgaben beteiligt. Die Landesanstalt für Umwelt, Messungen und Naturschutz stellt zusätzlich zu den Grundlagendaten auch die informationstechnische Infrastruktur zur Verfügung.

1.4 Das Räumliche Informations- und Planungssystem (RIPS)

RIPS ist die technische und organisatorische Querschnittskomponente des Umweltinformationssystems Baden-Württemberg zur Erfassung, Führung und Auswertung von Geodaten und wird vom Referat 53 der LUBW im Sachgebiet „Raumbezogene Informationssysteme“ betrieben. Zu den Hauptaufgaben zählen:

- *Bereitstellung von Geobasisdaten der Vermessungsverwaltung für die Fach- und Berichtskomponenten des UIS.*
- *Entwicklung und Betrieb effizienter Systeme zur Geodatenhaltung sowie die Bereitstellung von Geofunktionalitäten als Dienste für die einzelnen Fachanwendungen.*
- *Organisation des Geodaten-Austausch innerhalb der Verwaltung sowie die Geodatenabgabe an die Öffentlichkeit.*

Zum engeren Nutzerkreis gehören die LUBW, die Regierungspräsidien, die Landkreise und das Ministerium Ländlicher Raum. Weitere Informationen zum UIS und zum RIPS finden sich auf den Internetseiten der LUBW (LUBW_2006a).

2 GIS-Werkzeuge im RIPS-Umfeld

RIPS versteht sich als technisches, organisatorisches und fachliches Regelwerk, das neben einem Geodatenpool auch zahlreiche aufgabenabhängige GIS-Werkzeuge zur Verfügung stellt. Die folgende Tabelle gibt eine Übersicht über die Schwerpunkte des aktuellen Einsatzes:

Produkt	Technik	Entwicklung	Einsatzbereich	Bemerkung
RIPS-Viewer	Visual Basic MapObjects-LT	LfU/ITZ	Erfassungs- und Viewing-Komponente für shape-files; gut einsetzbar als CD-ROM-Viewer; einfach integrierbar in WAABIS-Übergangslösungen	Lizenzfreie Nutzung im UIS zur Prototypenentwicklung für Nutzeroberflächen und Funktionen von GISterm; Nutzung der ArcView-Datenformate
GISterm	JAVA	Fa. disy	Strategisches Produkt für die UIS-Berichts- und Fachsysteme im Internet/Intranet; Integration als Dienst in neuere UIS-Komponenten; multiuser-fähig	Als Produkt lizenzfrei im UIS nutzbar; laufende Erweiterung; Geodatenhaltung komplett in Oracle-DB
ArcView	AVENUE	Fa. ESRI	Desktop-GIS für single-user-Umgebungen; shape-file-basierte Geodatenhaltung	Relativ mächtiges Produkt für Kartographie und Analyse; Integration in andere Systeme aufwendig
ArcWaWiBo	AVENUE/VB	Fa. AHK	Mehr als 20 Zusatzfunktionen zum ArcView-Standard; ermöglicht Zugriff auf die zentrale Oracle-DB; wird als zentraler GIS- und Kartographie-Arbeitsplatz bei den UVB und RP eingesetzt	Entwicklung finanziert durch UVM; lizenzfreie Nutzung im UIS
Arc/Info	AML	Fa. ESRI	Umfangreiches Werkzeug für GIS-Spezialisten mit langjähriger Erfahrung	Komplettes GIS mit guter Methodendaten-bank; hohe Lizenz-kosten, Schulungs-aufwand erforderlich

ArcGIS 8.x/9.1	VBA	ESRI	Zentrale Geodatenverwaltung der landesweiten Geodatenbestände mit SDE. Aufbau von WebMapping-Services; wird auch zur Erfassung verschiedener UIS-Fachthemen eingesetzt	Besteht aus mehreren aufeinander abgestimmten Programmen wie ArcView, ArcInfo, ArcEditor, ArcSDE, ArcIMS
ArcGIS Server	.NET	ESRI	Ergänzung zu ArcIMS mit dem Ziel der Vereinheitlichung und Modularisierung der Webfunktionalität	Zugang erfolgt über Standardbrowser. Keine Zusatzsoftware oder aufwendige Softwareinstallationen erforderlich
ArcIMS	JavaScript PHP/HTML	ESRI	Zentraler Kartenservice „RIPS-Web“; Einsatz in Intranet und Internet als reines Auskunftssystem für eine große Nutzerzahl	

Tabelle 1: GIS-Werkzeuge im RIPS-Umfeld

Die Werkzeuge können in drei Gruppen eingeteilt werden. Zur Gruppe der Geoinformationssysteme zählen die ESRI-Produkte ArcView, das vorwiegend im Geschäftsbereich des Umweltministeriums eingesetzt wird, sowie ArcGIS/ArcInfo zur Datenorganisation und zur Analyse von Geodaten im RIPS-Pool. Daneben existiert zur Bereitstellung raumbezogener Umweltdaten ein auf ArcIMS basierender interaktiver Kartendienst „RipsWeb“. Dieser Service soll es einem breiten Anwenderkreis ermöglichen, über die gängigen (JavaScript-fähigen) Browser auf die Karten und Sachinformationen des UIS Baden-Württemberg zuzugreifen. Zu den Inhalten zählen unter anderem Umweltsachdaten zu Natur- und Landschaftsschutzgebieten, Naturparks und Wasserschutzgebieten. Neben ArcIMS-Diensten werden zukünftig vermehrt Anwendungen auf Basis von ArcGIS Server hinzukommen. Die dritte Gruppe bilden die speziellen Werkzeuge der einzelnen Fachanwendungen. Dazu gehören der RIPS-Viewer, GISterm und ArcWaWiBo aus dem WAABIS Modul 10 (LUBW_2006b).

2.1 Die ArcGIS-Produktfamilie

ArcGIS ist eine Produktfamilie aufeinander abgestimmter Werkzeuge zur Entwicklung von geographischen Informationssystemen (Abbildung 1).

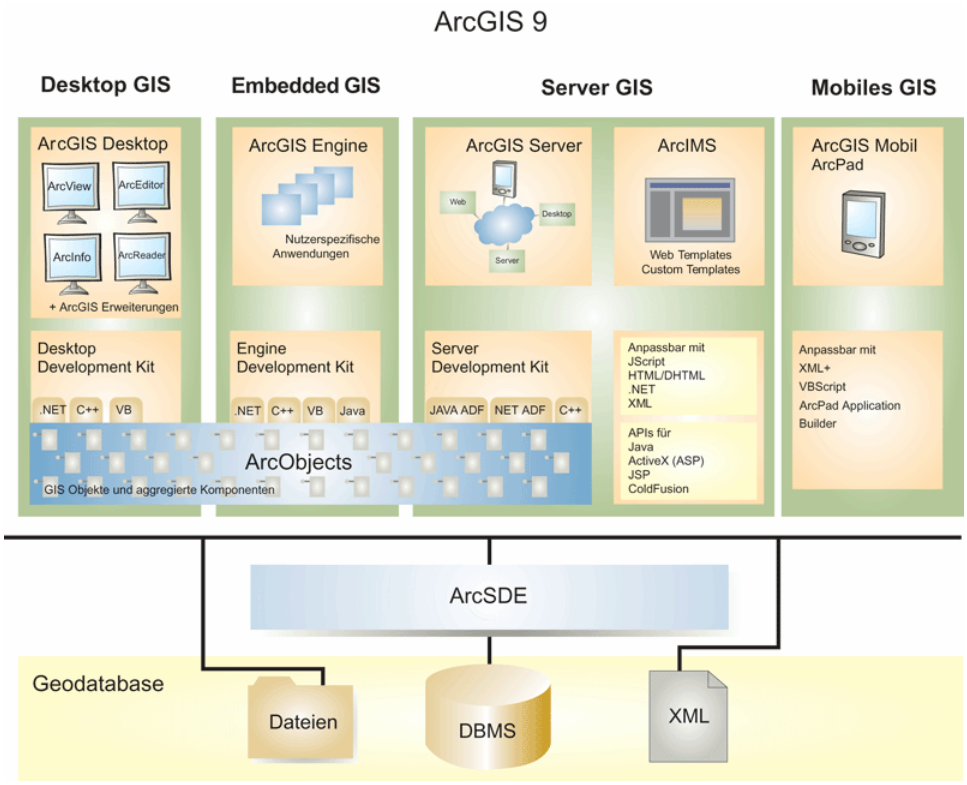

Abbildung 1: Die ArcGIS 9-Produktfamilie

ArcGIS unterscheidet zwischen Server-Diensten und Klienten. Die Server-Dienste ArcSDE (Datenbankgateway), ArcGIS Server (GIS-Funktionsserver) und ArcIMS (Internet Map Server) stellen verschiedene Grundfunktionalitäten bereit. Als Client-Applikationen sind in erster Linie ArcView, ArcEditor und ArcInfo zu nennen, die über eine integrierte Programmierumgebung mit Zusatzfunktionen erweitert werden können. Zwar werden auch weiterhin dateibasierte Datenquellen unterstützt, die ArcGIS-Strategie sieht jedoch vor, dass zukünftig alle Daten in objektrelationalen Datenbanken gespeichert und über das XML-Format ausgetauscht werden können (ESRI_2006a).

2.2 RIPS-Viewer

Der RIPS-Viewer ist ein Tool zur Visualisierung und Abfrage von Geodaten im UIS und wird vom Informationstechnischen Zentrum (ITZ) der LUBW im Sachgebiet 53.2 unter Visual Basic auf Basis der MapObjects-LT-Komponenten entwickelt. Es wird zwischen dem dem „kleinen“ und den „großen“ RIPS-Viewer unterschieden. Beim „kleinen“ RIPS-Viewer (Abbildung 2) handelt es sich um einen Geo-Dienst, der zur Orientierung in der Karte und zur Erfassung räumlicher Informationen in die verschiedenen UIS-Fachanwendungen integriert werden kann. Technisch gesehen ist diese Version kein eigenständiges Programm, sondern besteht aus ActiveX-Komponenten, die zusammen mit der Fachanwendung installiert werden.

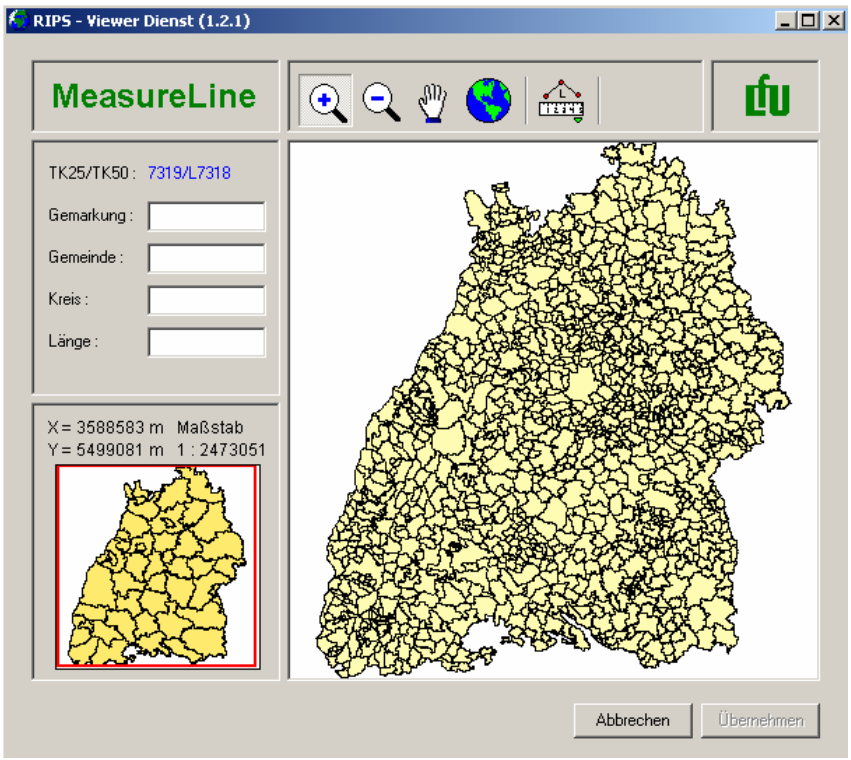

Abbildung 2: Der „kleine“ RIPS-Viewer

Im Gegensatz zum RIPS-Viewer Dienst ist der „große“ RIPS-Viewer (Abbildung 3) eine eigenständige Anwendung und kann unabhängig von der WAABIS-Umgebung genutzt werden. So kann der Anwender Abfragen auf Fach- und Geobasisdaten vordefinierter Karten ausführen, ohne dass zwingend eine Verbindung zur Datenbank bestehen

muss. Der „große“ RIPS-Viewer ist sowohl als einfaches Auskunftssystem als auch für komplexere GIS-Operationen einsetzbar.

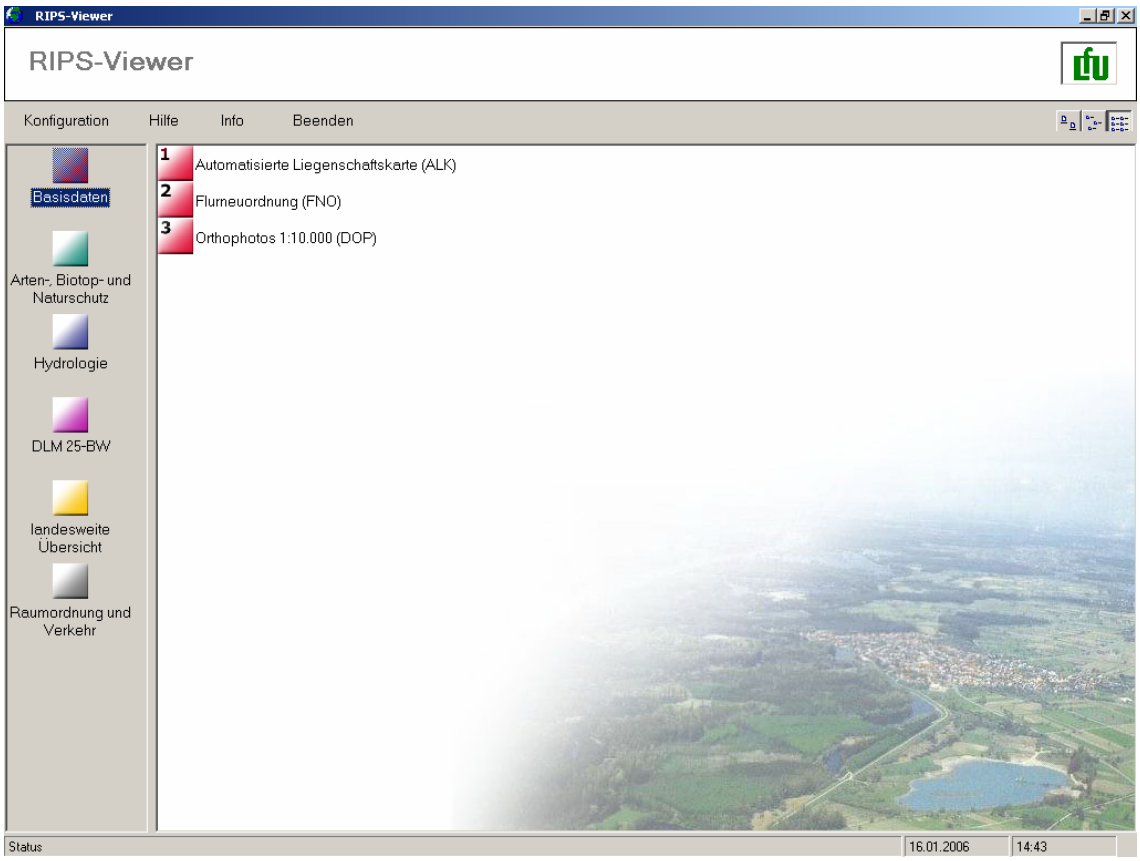

Abbildung 3: Der „große“ RIPS-Viewer

2.3 ArcView/ArcWaWiBo

Das Programm wird seit 1995 von der Firma AHK (Gesellschaft für Angewandte Hydrologie und Kartographie mbH) in Freiburg im Auftrag des Ministeriums für Umwelt und Verkehr Baden-Württemberg entwickelt. Es stellt dem Anwender auf Basis von ESRI ArcView eine Anwendungsschale für den Bereich Altlasten, Abwasser, Grundwasser und Bodenschutz bereit. Mit dieser in Visual Basic geschriebenen Erweiterung stehen neben den kartographischen Funktionen von ArcView eine Reihe zusätzlicher Analysewerkzeuge für die Pflege und Analyse von Geodaten zur Verfügung. Die Werkzeug-Palette von ArcWaWiBo ist in fünf Kategorien gegliedert (Abbildung 4). Die „Basic“-Tools beinhalten allgemeine Werkzeuge, die sich auf

nahezu alle Themen anwenden lassen. So bietet diese Funktionsgruppe z.B. die Möglichkeit, die Darstellung von Objekten zu ändern oder Geometrien miteinander zu verbinden. Die WAABIS-Tools stellen die Kommunikationsschnittstelle für die WAABIS-Datenbank und steuern die Generierung der Themenbeschriftung anhand der Vorgaben aus dem UIS. Mit den Erweiterungen „ALK"-Tools und „Karto"-Tools gibt es darüber hinaus Funktionalitäten für den Umgang mit der automatisierten Liegenschaftskarte und für die Erstellung von Karten.

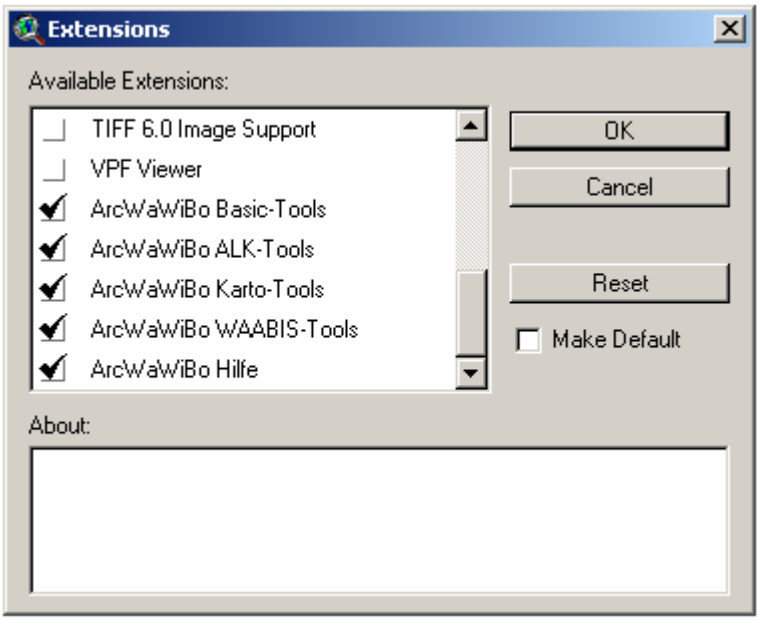

Abbildung 4: ArcWaWiBo-Extensions in ArcView

2.4 GISterm

GISterm (Abbildung 5) ist ein geographisches Informationssystem zur Visualisierung, Analyse und Erfassung von raumbezogenen Daten. Die Java-basierte Software wird von der Firma disy Informationssysteme in Karlsruhe entwickelt und ist als Geo-Dienst Bestandteil des zentralen UIS-Berichtssystems. Zudem wird sie als Komponente in verschiedenen Fachmodulen der UIS/WAABIS-Auslieferung sowie als eigenständige Anwendung eingesetzt. GISterm stellt umfangreiche Funktionen zur Erzeugung von Diagrammen und Kartogrammen bereit und wird vorwiegend für die Erfassung von Geodaten bzw. für die Erstellung thematischer Karten genutzt.

Um den Anforderungen dieser Nutzungsszenarien gerecht zu werden, mussten verschiedene Softwarekonzepte in die Entwicklung von GISterm einfließen. Die Bedeutung der Interoperabilität zu verschiedenen Datenquellen stellen hohe Ansprüche an die Skalierbarkeit der Anwendung. Die Software muss unabhängig vom verwendeten Betriebssystem arbeiten und darüber hinaus einfach in Web-Anwendungen integrierbar sein. Durch den Einsatz so genannter Datenquellen-Operatoren wird gewährleistet, dass GISterm mit bereits vorhandenen wie auch mit zukünftigen Datenquellen umgehen kann. Die Plattformunabhängigkeit ergibt sich aus der Implementierung in der Programmiersprache Java. Weiterhin wurde das Konzept der mehrschichtigen Architektur verwirklicht. GISterm besteht aus einem Client, einem Application Server und einem Daten Server und kann dadurch sowohl als eigenständige Applikation wie auch im Rahmen einer Inter-/Intranet-Umgebung genutzt werden. Als drittes wesentliches Konzept ist die komponentenbasierte Entwicklung zu nennen. Die Teilmodule von GISterm wurden als JavaBeans realisiert und können daher problemlos in anderen Softwarekomponentenszenarien eingesetzt werden (GISTERM_2006).

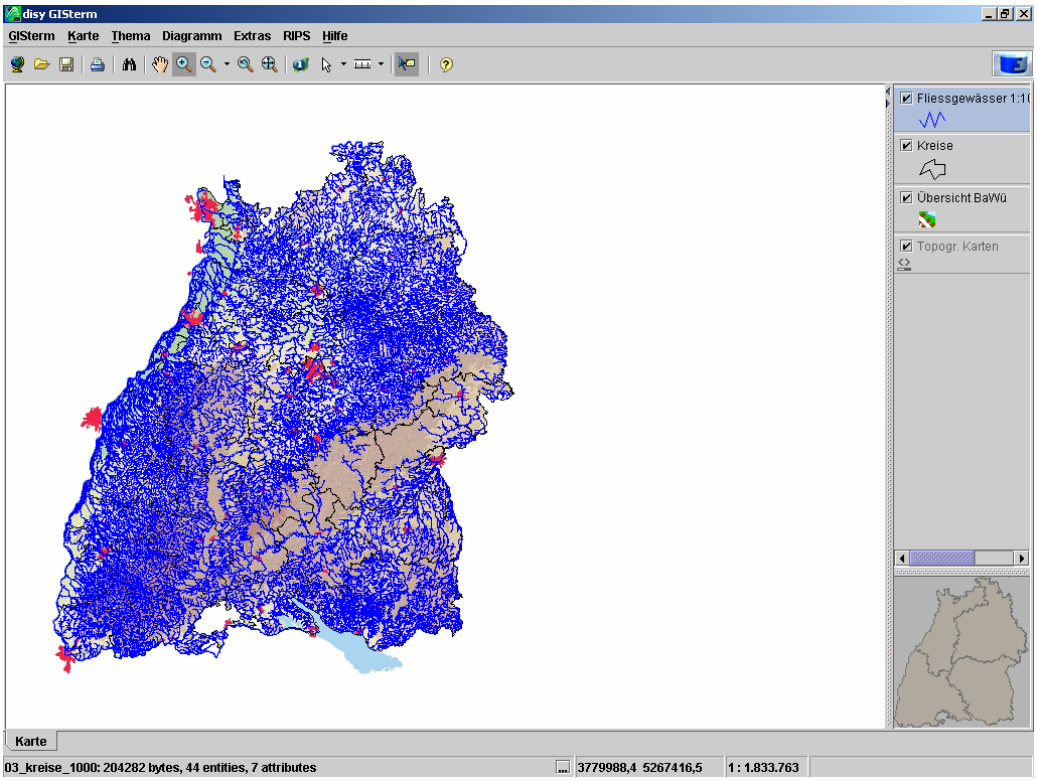

Abbildung 5: GISterm der Firma disy Informationssysteme

3 Technische Grundlagen

3.1 XML Web Services

Web Services bilden die grundlegenden Bausteine für die Verwirklichung des verteilten Computing im Internet und werden nach Dustdar et al (2003) häufig als Beispiel für eine Service-orientierte Architektur angeführt. Web Services sind im Gegensatz zu Web-Applikationen nicht für die Interaktion mit einem menschlichen Nutzer vorgesehen, sondern dienen in erster Linie dem Datenaustausch zwischen Softwaresystemen. Abbildung 6 verdeutlicht die Funktionsweise der Web Service Infrastruktur, in der es drei wesentliche Komponenten gibt: Service-Anbieter, Service-Broker und Service-Konsument. Zunächst veröffentlicht der Service-Anbieter (Service Provider) eine Beschreibung seines Dienstes in einem Verzeichnis (Service Broker). Diese Registrierungsstelle kann vom Konsumenten (Service Requestor) durchsucht werden. Nachdem dieser einen Dienst ausgewählt hat, erfolgt die dynamische Anbindung an den Provider, so dass die Methoden des Web Service in der Applikation des Konsumenten verwendet werden können.

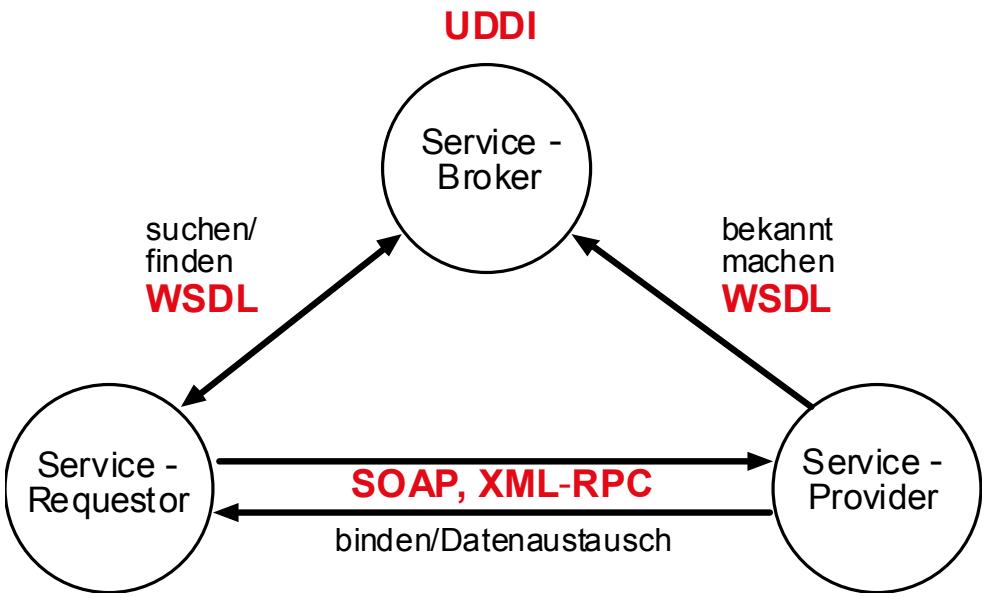

Abbildung 6: Funktionsweise von Web Services

Web Services sind über einen „Uniform Ressource Identifier“ (URI) eindeutig identifizierbar und können über diese Zeichenfolge im Internet angesprochen werden. Die eXtensible Markup Language (XML) bildet die technische Basis sämtlicher Formate und Protokolle der Web Service-Architektur, allerdings sind bislang noch nicht alle Vorgänge innerhalb der Kommunikation standardisiert. So gibt es beispielsweise noch offene Fragen bezüglich der Dienstesicherheit, der Transaktionen und der Service-Qualität, die zukünftig geklärt werden müssen. Weitgehende Übereinstimmung unter den Softwareherstellern herrscht dagegen bei den Basistechnologien SOAP (Simple Object Access Protocol), WSDL (Web Service Description Language) und UDDI (Universal Description, Discovery and Integration).

3.1.1 Simple Object Access Protocol (SOAP)

Bei SOAP handelt es sich um ein relativ einfaches Protokoll zur Kommunikation und dem Austausch von Daten zwischen Web Services und deren Konsumenten. Der Standard legt die Regeln für die Übertragung von Nachrichten zwischen den Applikationen fest und eignet sich insbesondere für die Ausführung von Remote-Procedure-Calls (RPC). SOAP bedient sich der XML für die Darstellung der Daten, während der Transport über die bekannten Internetprotokolle wie HTTP/TCP erfolgt. SOAP-Nachrichten sind nach dem Head-Body-Pattern strukturiert und haben den in Abbildung 7 dargestellten Aufbau. Der Envelope-Container besteht aus einem optionalen SOAP-Header, einem Body-Block sowie einem festgelegten Namensraum. Im Header-Block sind Metadaten abgelegt, die unter anderem Informationen zur Authentifizierung oder zur Zugehörigkeit zu einer Transaktion enthalten können. Der Body-Block ist unbedingt notwendig und enthält die eigentlichen Nutzdaten der Nachricht, die beispielsweise für entfernte Methodenaufrufe stehen können. Nachfolgend sind je ein Beispiel für eine SOAP-Anfrage sowie für eine SOAP-Antwort eines Servers samt HTTP-Header zu sehen:

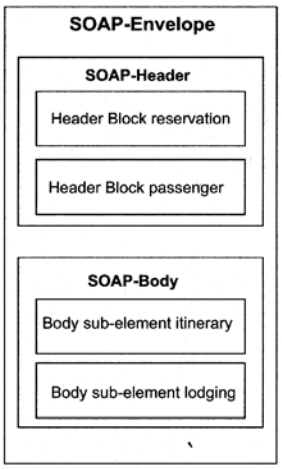

Abbildung 7: Struktur einer SOAP-Nachricht

SOAP-Anfrage:

```
POST /RipsWebService2/LinearReferencing/Measure.asmx HTTP/1.1
Host: localhost
Content-Type: text/xml; charset=utf-8
Content-Length: length
SOAPAction: "http://tempuri.org/FindNearest(x,y,SearchRadius)"

<?xml version="1.0" encoding="utf-8"?>
<soap:Envelope xmlns:xsi="http://www.w3.org/2001/XMLSchema-
instance" xmlns:xsd="http://www.w3.org/2001/XMLSchema"
xmlns:soap="http://schemas.xmlsoap.org/soap/envelope/">
  <soap:Body>
    <FindNearest_x0028_x_x002C_y_x002C_SearchRadius_x0029_
xmlns="http://tempuri.org/">
      <p_dX>double</p_dX>
      <p_dY>double</p_dY>
      <p_dSearchRadius>double</p_dSearchRadius>
      <p_wsType>Access or Raster or Shape or Sde</p_wsType>
      <p_strFeatClass>string</p_strFeatClass>
    </FindNearest_x0028_x_x002C_y_x002C_SearchRadius_x0029_>
  </soap:Body>
</soap:Envelope>
```

SOAP-Antwort:

```
HTTP/1.1 200 OK
Content-Type: text/xml; charset=utf-8
Content-Length: length

<?xml version="1.0" encoding="utf-8"?>
<soap:Envelope xmlns:xsi="http://www.w3.org/2001/XMLSchema-
instance" xmlns:xsd="http://www.w3.org/2001/XMLSchema"
xmlns:soap="http://schemas.xmlsoap.org/soap/envelope/">
  <soap:Body>

<FindNearest_x0028_x_x002C_y_x002C_SearchRadius_x0029_Response
xmlns="http://tempuri.org/">

<FindNearest_x005F_x0028_x_x005F_x002C_y_x005F_x002C_SearchRadi
us_x005F_x0029_Result>
        <MeasureVal>
          <m_featClass>string</m_featClass>
          <m_xCoord>double</m_xCoord>
          <m_yCoord>double</m_yCoord>
          <m_mValue>double</m_mValue>
          <m_objectID />
        </MeasureVal>
        <MeasureVal>
          <m_featClass>string</m_featClass>
          <m_xCoord>double</m_xCoord>
          <m_yCoord>double</m_yCoord>
          <m_mValue>double</m_mValue>
```

```
            <m_objectID />
          </MeasureVal>

    </FindNearest_x005F_x0028_x_x005F_x002C_y_x005F_x002C_SearchRad
    ius_x005F_x0029_Result>

    </FindNearest_x0028_x_x002C_y_x002C_SearchRadius_x0029_Response
    >
      </soap:Body>
    </soap:Envelope>
```

3.1.2 Web Service Description Language (WSDL)

Die WSDL ist eine XML-basierte Metasprache zur Beschreibung der Schnittstellen eines Web Service. Das WSDL-Dokument wird dem Service-Konsumenten über den Broker bereitgestellt und ist die Grundlage eines Vertrages, der dem Client mitteilt, wie sich der XML-Dienst genau verhält. Ziel der WSDL ist es, einen von Maschinen lesbaren Zugang zu Web Services zu schaffen, so dass unterschiedliche Clients automatisch verstehen, wie sie mit diesen kommunizieren können. Dadurch soll eine dynamische und zur Laufzeit eines Systems stattfindende Integration von Diensten zu neuen Anwendungen möglich sein (Kuschke & Wölfel, 2002). WSDL-Schema-Dokumente beinhalten eine Spezifikation sowohl von den Methodenschnittstellen als auch von den sonstigen technischen Daten zum Auffinden eines Web Service und zur korrekten Abwicklung der Kommunikation. Diese Spezifikation besteht aus einem abstrakten und einem konkreten Teil (Abbildung 8). Der abstrakte Teil definiert sprach- und plattformabhängige Typen, Nachrichten, Operationen und Porttypen, während der konkrete Teil den eigentlichen Kommunikationsvorgang (z.B. die Serialisierung von Objekten) durchführt. Diese Trennung ist sinnvoll, um die Struktur von Web Services von der Kommunikation unabhängig zu halten.

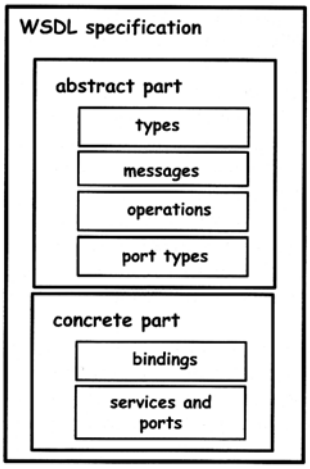

Abbildung 8: WSDL-Spezifikation

Abstrakter Teil:

- *Nachrichten (message):*

  Abstrakte Definitionen der Nutzdaten (Parameter einer Methode oder Rückgabewert einer Funktion).

- *Typen (types):*

  Datentypen, die zum Austauschen der Nachrichten verwendet werden.

- *Operationen (operations):*

  Verbinden zusammengehörige Messages zu einer logischen Einheit.

- *Porttyp (portType):*

  Fasst die Nachrichten zu abstrakten Operationen zusammen.

Konkreter Teil:

- *Bindungen (bindings):*

  Definieren die Nachrichtenformate und die Details, wie die PortType-Operationen mit dem jeweiligen Protokoll übertragen werden.

- *Services und Ports:*

  Ports kombinieren die Bindungen mit einem URI, während Services eine Aggregation mehrerer Ports darstellen.

Aufbau eines WSDL-Dokuments

```
<definitions>
  <types> </types>
  <message> </message>
  <portType> </portType>
  <binding> </binding>
  <service> </service>
</definitions>
```

WSDL-Dokumente sind zwar recht einfach aufgebaut, können aber mit zunehmender Komplexität der Interaktionen, welche sie beschreiben, sehr unübersichtlich werden. Visual Studio bietet daher eine automatisierte Erstellung des WSDL-Dokuments, so dass sich der Entwickler in der Regel nicht mit den genauen Details beschäftigen muss.

3.1.3 Universal Description, Discovery and Integration (UDDI)

UDDI ist ein Verzeichnisdienst zum Beschreiben, Auffinden und Integrieren von Web Services in die eigene Applikation. Es handelt sich dabei nicht um eine konkrete Implementierung, diese ist den jeweiligen Anbietern überlassen. Vielmehr stellt UDDI ein Framework dar (Abbildung 9), das aus den drei Komponenten „Business“, „Service“ und „Technische Spezifikationen“ besteht (Dustdar et al, 2003). Der Business-Teil enthält alle geschäftsrelevanten Informationen wie Firmenname, Kontaktadressen usw. Im Bereich Service sind die Metadaten zu Web Services abgelegt, die von den jeweiligen Unternehmen angeboten werden, während sich in den Spezifikationen alle technischen Details über den Dienst befinden. Dazu gehört unter anderem das im vorangegangen Kapitel erläuterte WSDL-Dokument.

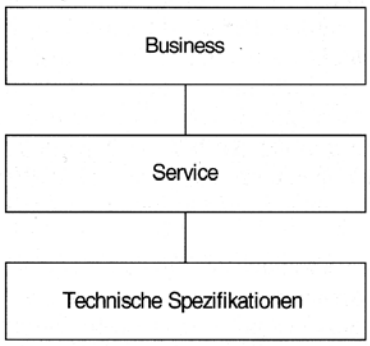

Abbildung 9: Das UDDI-Framework

3.2 C# und das Microsoft .NET-Framework

3.2.1 Microsoft .NET

.NET ist nicht nur eine neue Programmierplattform, sondern vielmehr Microsofts Strategie, Informationen und Dienste jederzeit auf Basis der Web Service-Technologie verfügbar zu machen und damit die Anwendungsentwicklung in verteilten Umgebungen zu vereinfachen. Die beiden zentralen Bestandteile des Systems bilden die Common Language Runtime (CLR) sowie die umfangreiche Klassenbibliothek. Das .NET-

Framework stellt mehrere Compiler zur Verfügung und unterstützt dadurch den Einsatz einer ganzen Reihe von Programmiersprachen wie C#, Visual Basic .NET, Java und C++ (Abbildung 10). Diese Compiler erzeugen einen Zwischencode, die so genannte Microsoft Intermediate Language (MSIL), der erst zur Laufzeit der Anwendung von der CLR in prozessorspezifischen Maschinencode umgewandelt wird. Die Klassenbibliothek bietet eine umfangreiche Sammlung wieder verwendbarer, objektorientierter Funktionen für den Einsatz in praktisch allen Bereichen. Dabei nutzen sämtliche .NET-Sprachen ein- und dieselbe Bibliothek, so dass die Wahl der Programmiersprache zweitrangig wird. Dadurch ist selbst innerhalb eines einzelnen Projektes der Einsatz unterschiedlicher Sprachen möglich (Doberanz & Kowalski, 2003).

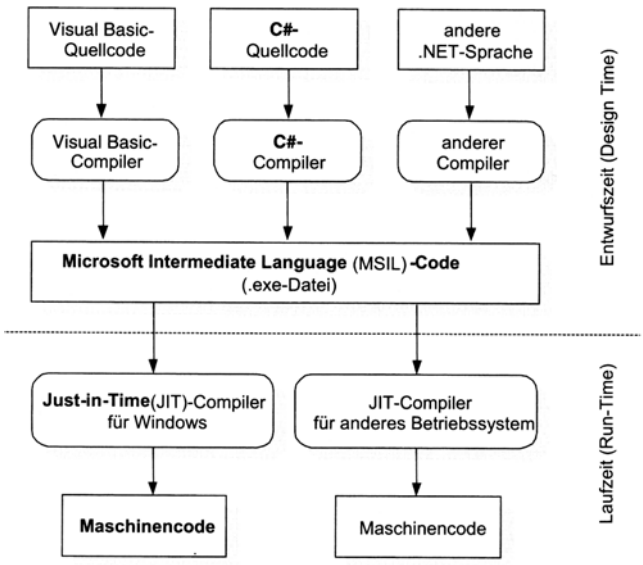

Abbildung 10: Architektur von .NET

3.2.2 Die Sprache C#

C# ist die Systemsprache des .NET-Frameworks und wurde ausschließlich für diese Plattform konzipiert. Daher konnten die Entwickler auf Kompromisse, wie sie für andere .NET-Sprachen eingegangen werden mussten, verzichten. Es wurde der Versuch unternommen, das Beste aus den etablierten Programmiersprachen wie Java, JavaScript, Visual Basic und C++ zu kombinieren, ohne aber deren Nachteile zu übernehmen. So wurde beispielsweise die automatische Speicherbereinigung

(Garbage Collection) von Java übernommen, wohingegen auf die Zeiger-Technik verzichtet wurde. C# erfüllt als konsequent objektorientierte Sprache die Kriterien der Abstraktion, der Kapselung, Polymorphie und Vererbung (Doberanz & Kowalski, 2003).

3.2.3 Microsoft Visual Studio 2003 .NET

Visual Studio 2003. NET ist eine integrierte Entwicklungsumgebung mit Modulen für die Programmierung mit Visual Basic, C, C++, C# und J++. Die Software wird als Standard-, Professional- oder Enterprise-Version ausgeliefert. Visual Studio beinhaltet eine erweiterte Version des .NET-Frameworks 1.1 und basiert vollständig auf der Integration von XML-Webdiensten. Dadurch können Anwendungen aus dem Internet unabhängig von der Plattform, von der Programmiersprache oder dem Objektmodell, in der sie erstellt wurden, in Visual Studio zu neuen Systemen zusammengesetzt werden. Ende 2005 brachte Microsoft den Nachfolger Visual Studio 2005 auf den Markt (MSVSTUDIO_2005).

3.2.4 Weshalb .NET für das RIPS-Framework?

Die Anforderungen an die moderne Softwareentwicklung haben sich in den letzten Jahren durch den wachsenden Einfluss des World Wide Web stark gewandelt. Durch die zunehmende Bedeutung verteilter Systeme werden die Ansprüche an die Skalierbarkeit einer Anwendung, die Verteilbarkeit auf mehrere Schichten sowie die Aspekte der Sicherheit immer größer. Nach Wigard (2003) wurde das .NET-Framework speziell dafür konzipiert, kommerzielle Anwendungen im Internet-Umfeld zu entwickeln. Mit ASP.NET und der Entwicklungsumgebung Visual Studio .NET stellt Microsoft ein komfortables Programmiersystem für die Erstellung dynamischer Webanwendungen und XML Web Services zur Verfügung. Neben den bisher genannten Vorteilen wie der freien Wahl der Programmiersprache, der Unterstützung für verteilte Systeme und der umfangreichen Klassenbibliothek führen Semmler und Apfel (2002) die Plattformunabhängigkeit und die Konformität zu weitverbreitenden Standards als wesentliche Aspekte an. .NET integriert Basistechnologien wie XML, SOAP, UDDI und TCP/IP und garantiert damit eine breite Akzeptanz unter den Anwendern. Außerdem können bestehende COM-Objekte durch den Einsatz spezieller Hüllklassen (so

genannter „Wrapper“) in .NET-Anwendungen integriert werden, wie es auch umgekehrt möglich ist, .NET-Assemblies in COM einzusetzen. Durch diese Interoperabilität müssen alte Komponenten nicht erst umständlich portiert, sondern können direkt in der neuen Umgebung verwendet werden. Im Hinblick auf das RIPS Web Service Framework ist nicht zuletzt zu erwähnen, dass die ArcGIS Server ADF vollständig auf dem .NET-Framework aufsetzt. (vgl. Kapitel 3.3).

3.3 ESRI ArcGIS Server 9.0

3.3.1 Systemarchitektur

Der ESRI ArcGIS Server ist eine Plattform, um unternehmensweite, Server-seitige GIS-Infrastrukturen aufzubauen (ESRI, 2004). Dabei werden die GIS-Funktionalitäten von zentralen Servern bereitgestellt und können von verteilten Anwendern genutzt werden. Mit dem ArcGIS Server Application Developer Framework (ADF) können unter anderem Web-Applikationen, Web Services und Desktop-Applikationen, die mit einem Server interagieren, erstellt werden. Wie auf Abbildung 11 zu sehen ist, besteht das System aus mehreren Komponenten:

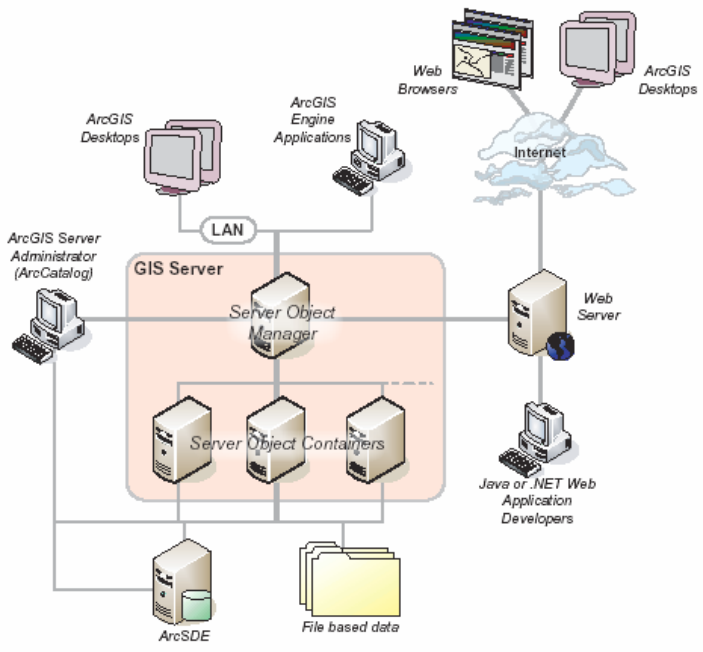

Abbildung 11: Systemarchitektur des ArcGIS Server

GIS Server

Der GIS Server ist verantwortlich für das Hosting und die Verwaltung der Server Objects, die von den Web Services, Web- oder Desktop-Applikationen verwendet werden können. Bei den Server Objects handelt es sich um Komponenten der ArcObjects-Bibliothek. Der GIS Server besteht aus dem Server Object Manager (SOM) und einem oder mehrerer Server Object Container (SOC):

- *Server Object Manager*

  Ein Windows-Dienst eines einzelnen Rechners, der die Server Objects verwaltet. Baut eine Applikation oder ein Service eine Verbindung zum Server auf, wird immer der SOM angesprochen.

- *Server Object Container*

  Diese Container hosten die Server Objects, die vom SOM verwaltet werden. Die Server Objects werden innerhalb eines Container-Prozesses ausgeführt, wobei ein Prozess bis zu vier Server Objects verwalten kann. Ebenso kann der Container mehrere Container-Prozesse hosten.

- *Server Directory*

  Hier werden die temporären Image-Dateien gespeichert, die von den Server Objects für die Verwendung durch den Client erzeugt werden.

Web Server:

Auf dem Web Server werden Web-Applikationen und Web Services abgelegt, die mit der ArcGIS Server API erstellt wurden. Sie nutzen die API, um über den SOM eine Verbindung zu den Server Objects herzustellen. Dieser liefert einen Proxy auf die angeforderten Objekte zurück, mit dem der Client schließlich arbeiten kann, als seien die Objekte Teil der Web Server-Applikation selbst. Dennoch wird die gesamte GIS-Funktionalität auf dem GIS Server ausgeführt. Der Web Server regelt lediglich die Kommunikation mit dem Browser und die Logik der Anwendung (Abbildung 12).

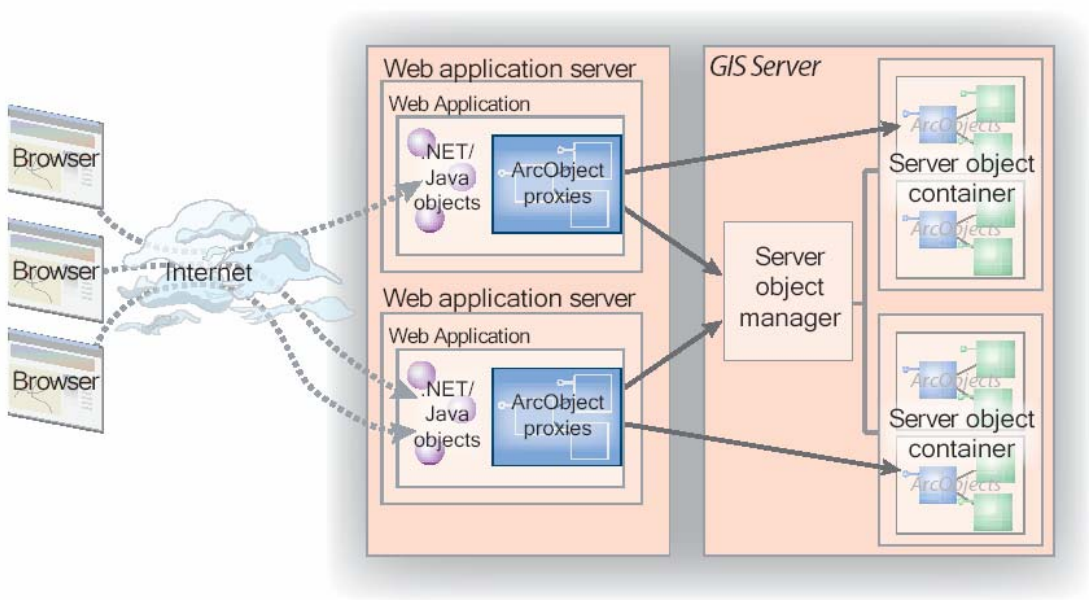

Abbildung 12: Funktionsweise des Web Servers

Client-Anwendungen

Web Browser können über HTTP Verbindung zu einer Web-Applikation auf dem Web Server aufnehmen, um die Funktionalität des GIS Servers zu nutzen. ArcGIS Desktop-Anwendungen greifen hierzu auf die verfügbaren Web Services zu. Innerhalb eines Local Area Network (LAN) können sie sich auch direkt mit dem GIS Server verbinden (Abbildung 11). Eine dieser Anwendungen ist ArcCatalog, die dazu verwendet wird, den GIS Server zu administrieren.

3.3.2 Server Objects

Die ArcObjects-Bibliothek bildet das technologische Basis-Framework aller ArcGIS Komponenten. Auch Server Objects sind ArcObjects und stellen Geodaten zur Verarbeitung bereit. ArcGIS Server unterscheidet prinzipiell zwei Typen von Server Objects:

- *GeocodeServer Objects (Location-Library)*
- *MapServer Objects (Carto-Library)*

Sie werden auch als grobkörnige Komponenten bezeichnet, hinter denen sich wiederum feinkörnige ArcObjects befinden, über die auf die einzelnen Elemente einer Map oder eines Locators zugegriffen werden kann. Abbildung 13 zeigt den Aufbau eines MapServer Objects.

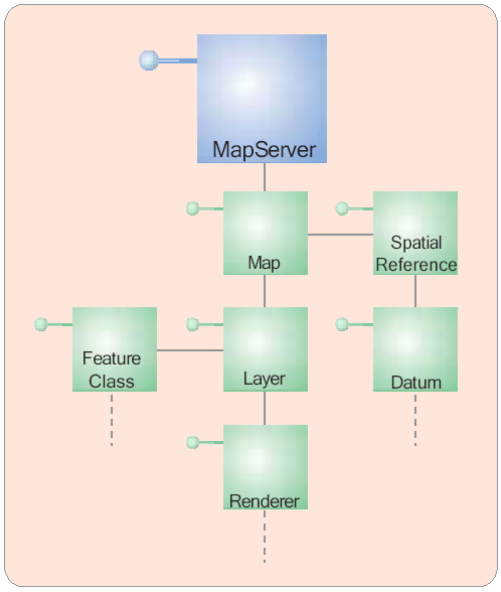

Abbildung 13: Aufbau eines MapServer Objects

Server Objects können über eine eingebaute SOAP-Schnittstelle selbst als Web Service bereitgestellt und so direkt über Internet oder Intranet von Client-Anwendungen verwendet werden. Im Gegensatz zu ArcObjects können Server Objects vom Administrator des GIS Servers vorkonfiguriert werden. Dabei sind folgende Parameter zu definieren:

- *Name des Server Objects*
- *Typ des Server Objects*

  d.h. MapServer oder GeocodeServer. Der Typ bestimmt die Initialisierungsparameter sowie die Methoden und Eigenschaften, die dem Entwickler zur Verfügung stehen.

- *Initialisierungsdaten und –Parameter*

  Wird ein MapServer Object erzeugt, wird dieses mit dessen Map-Dokument (einer mxd-Datei) initialisiert. Ein GeocodeServer Object wird dagegen mit dem entsprechenden Address Locator initialisiert (z.B. einer loc-Datei).

- *Server Object Pooling*

  Server Objects können als „pooled“ oder „unpooled“ deklariert werden. Wird die Option „unpooled“ gewählt, so wird für jede Anfrage eines Klienten eine neue Instanz des Server Objects erzeugt und wieder zerstört, sobald sie nicht mehr benötigt wird. Dadurch ist die Zahl der unpooled Instanzen immer 1:1 zur Anzahl der Clients des Objektes. Wird dagegen die Option „pooled“ gewählt, wird sofort beim Start des Server Objects die bei der Konfiguration festgelegte Minimalzahl an Instanzen aufgebaut und in den Arbeitsspeicher geladen. Pooled Server Objects sind im Vergleich zu den unpooled leistungsfähiger, da sie den Client-Anwendungen direkt aus dem Arbeitsspeicher bereitgestellt werden können. Unpooled Server Objects beanspruchen sehr viel Zeit bei der Erzeugung, belasten dafür aber nicht den Arbeitsspeicher des Systems.

- *Server Object Isolation*

  Server Objects werden innerhalb von Prozessen auf den Containern ausgeführt. Objekte mit hoher Isolation teilen keine Prozesse mit anderen Server Objects, sondern starten jeweils ihren eigenen. Server Objects mit niedriger Isolation können dagegen Prozesse mit bis zu vier anderen Objekten desselben Typs teilen. Ein hoher Isolationsgrad beansprucht mehr Ressourcen und führt schnell zur Auslastung des Systems, hat aber den Vorteil, dass im Falle eines Fehlers nur dieses eine Objekt betroffen ist.

- *Server Object Recycling*

  Server Objects können unbrauchbar werden, wenn Client-Anwendungen Änderungen am Zustand eines Objektes vornehmen oder fälschlicherweise Referenzen auf Objekte halten, die deren Verwendung durch andere Clients

blockieren. Das Setzen der Option „Recycling“ erlaubt dem Server das Austauschen solcher „korrupter“ Objekte und die Freigabe von belegten Ressourcen.

3.3.3 Das Application Developer Framework (ADF) für .NET

Neben dem GIS Server zur Bereitstellung und Verwaltung der Geodaten bildet das Application Developer Framework die zweite Hauptkomponente des ArcGIS Server. Dieses Programmiersystem ist auf ArcObjects und das Microsoft .NET-Framework aufgesetzt und erweitert diese Bibliotheken um zahlreiche neue Klassen und Schnittstellen, welche die Integration von GIS-Funktionalitäten in die eigene Web-Applikation ermöglichen (Abbildung 14). Nach der Installation des ADF stehen in Visual Studio. NET zusätzliche Templates zur Erstellung verschiedener Kartendienste sowie die ASP.NET WebControls-Assembly zur Verfügung, deren Objekte die Interaktion mit dem GIS Server ermöglichen.

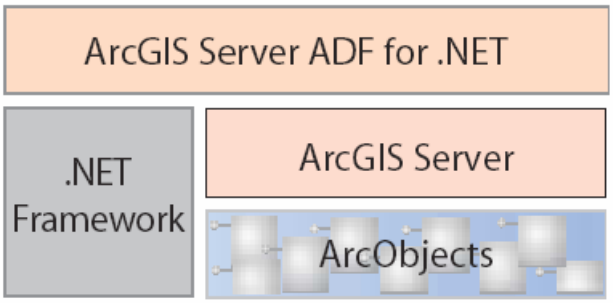

Abbildung 14: Das ArcGIS Server ADF für .NET

4 Erstellung ausgewählter Framework-Komponenten

4.1 Die Testfunktionen

4.1.1 Measure-Funktionalität

Die Klasse stellt zwei Überladungen zur M-Wertbestimmung bereit:

- *FindNearest(x, y, searchRadius)*
- *FindNearest(x, y, objectID)*

Die erste der beiden Methoden erhält eine Koordinate und einen Suchradius und bestimmt anhand dieser Parameter den Measure-Wert auf dem zum Eingabepunkt nächsten Gewässer. Durch eine räumliche Verschneidung werden alle Features innerhalb des Suchradius selektiert, worauf mit Hilfe der *ReturnDistance*-Methode des *IProximityOperator*-Interface das nächste Gewässer bestimmt werden kann. Nachfolgend ist die Methode *GetNearestPointOnPolyline* zu sehen. Sie implementiert *ICurve::QueryPointAndDistance* zur Bestimmung des am nächsten liegenden Punktes auf dem Objekt (samt M-Wert):

```
private IPoint GetNearestPointOnPolyline(IPoint p_location,
  IFeature p_nearestFeature)
{
  IPolyline l_polyline;
  IPoint l_outPt;
  double l_distAlongCurve = 0;
  double l_distFromCurve = 0;
  Boolean l_bRight = false;
  l_outPt = new PointClass();
  l_polyline = (IPolyline) p_nearestFeature.Shape;

  l_polyline.QueryPointAndDistance(
    ESRI.ArcGIS.Geometry.esriSegmentExtension.esriNoExtension,
    p_location, false, l_outPt, ref l_distAlongCurve, ref
    l_distFromCurve, ref l_bRight);

  return l_outPt;
}
```

Bei der zweiten Überladung hat der Nutzer die Möglichkeit, direkt ein Gewässer anzugeben, auf dem der M-Wert ermittelt werden soll. Das ist nützlich, wenn beispielsweise der direkte Abfluss zum nächsten Gewässer durch ein anderes Objekt, etwa einen Berg, versperrt ist. Abbildung 15 zeigt die Maske zur Eingabe der Parameter in ArcMap.

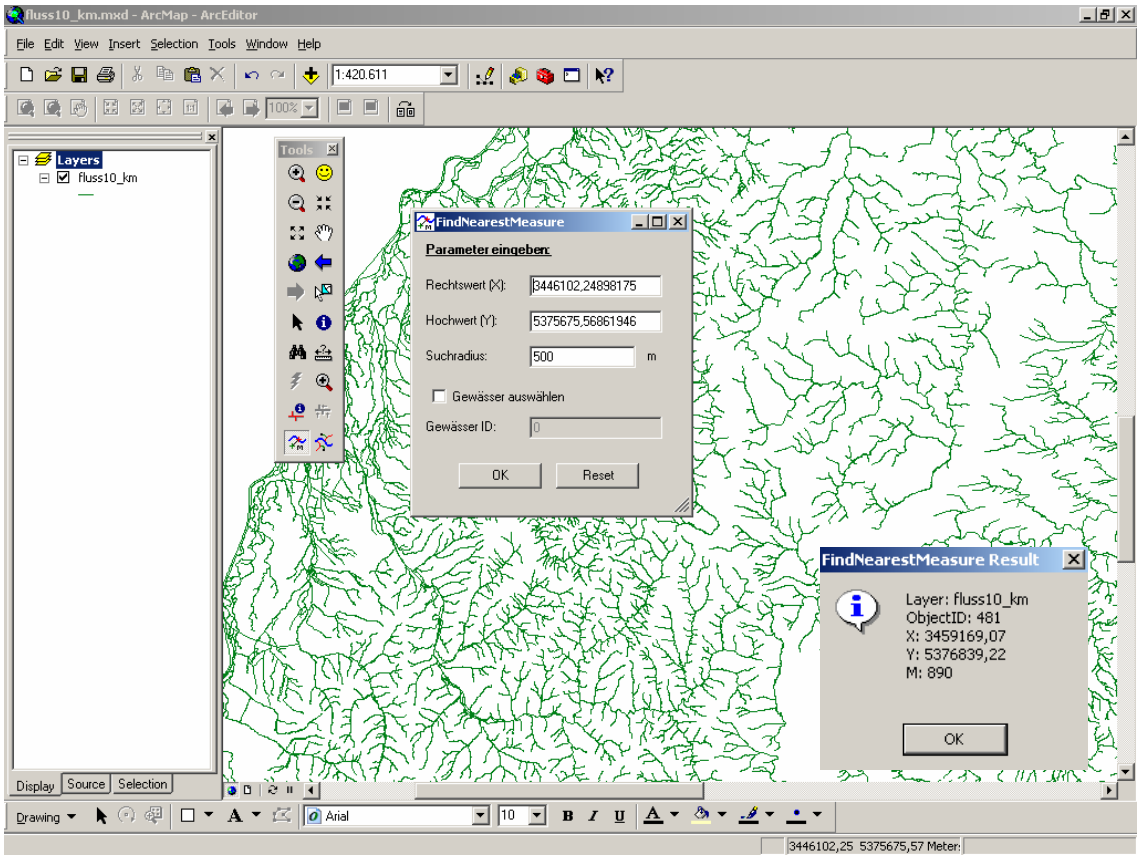

Abbildung 15: M-Wertbestimmung in ArcMap

4.1.2 Stream-Funktionalität

Diese Klasse stellt zwei Methoden zur Bestimmung der Vorfluter eines Fließgewässers bereit. Ähnlich wie bei der M-Wertbestimmung wird das zu einem Eingabepunkt nächste Gewässer gewählt oder der Anwender übergibt eine Objekt-ID, um direkt ein Gewässer zu wählen. Jedes Fließgewässer besitzt ein Feld „Vorfluter“, in dem die Gewässerkennzahl (GKZ) des Vorfluters abgelegt ist. Die Funktion ermittelt in einer Schleife zunächst diese GKZ. Ist deren Wert ungleich 0, wird der Vorfluter aus der

FeatureClass extrahiert und einer Liste hinzugefügt. Die Schleife bricht ab, sobald die Vorfluter-Kennzahl 0 ist:

```
private ArrayList GetReceiver(IFeature p_feat, string
  p_strFeatClass)
{
  ArrayList l_receiverList = new ArrayList();
  double l_receiverID = 0;

  // Die Schleife wird so lange ausgeführt, bis
  // das Gewässer keinen Vorfluter mehr besitzt
  do
  {
    l_receiverID = Convert.ToDouble((p_feat.get_Value(
      p_feat.Fields.FindField("VORFLUTER"))));

    if (l_receiverID != 0)
    {
      // Extrahiere den Vorfluter aus der FeatureClass
      // und füge ihn der ArrayList hinzu
      string l_whereClause = "GKZ = " +
l_receiverID.ToString();
      IFeature l_receiver = GetFeature(p_strFeatClass,
        l_whereClause);
      l_receiverList.Add(l_receiver);
      p_feat = l_receiver;
    }
  }
  while(l_receiverID != 0);

  return l_receiverList;
}
```

4.2 ArcGIS-Erweiterungen mit Visual C#

4.2.1 ArcObjects und .NET-Komponenten

Da ArcObjects auf der Microsoft COM-Technologie basieren, können .NET-Komponenten nicht unmittelbar mit ihnen interagieren. Um ArcGIS-Applikationen dennoch mit nutzerspezifischen .NET-Objekten erweitern zu können, müssen diese zunächst für COM registriert und deren Assemblies in eine Typbibliothek (tlb-Datei) exportiert werden. Das .NET-Framework stellt hierzu im Namespace *System.Runtime.InteropServices* den COM-Interop Service bereit, der es .NET ermöglicht, auf ArcObjects zuzugreifen, als seien diese selbst .NET-Komponenten.

Visual Studio übernimmt die Registrierung für COM automatisch, sofern der Entwickler die Option „Für COM-Interop registrieren“ in den Konfigurationseinstellungen auf „true“ setzt. Damit wird beim Kompilieren der Klasse neben der dll-Datei (Dynamic Link Library) auch eine tlb erzeugt (Abbildung 16).

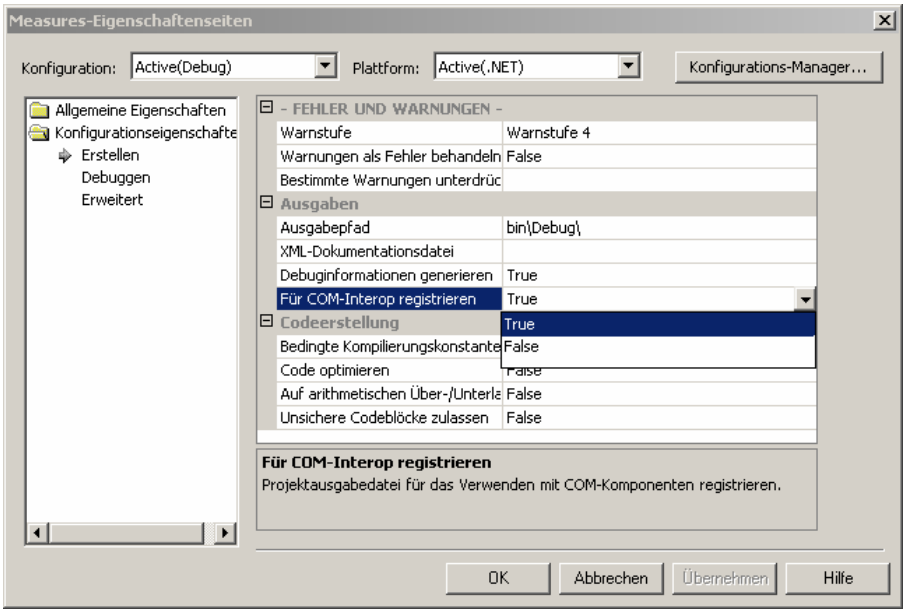

Abbildung 16: Registrierung einer .NET-Komponente für COM-Interop

Nachdem die Registrierung für COM erfolgt ist, müssen die .NET-Komponenten abhängig von ihrem Kontext in COM-Komponentenkategorien registriert werden. So müssen z.B. sämtliche ArcMap Commands und Tools für die MxCommands-Kategorie registriert sein. Auf die einfachste Weise funktioniert das, in dem der Klasse Code hinzugefügt wird, der die Komponente automatisch für die entsprechende Kategorie registriert, sobald sie für COM registriert wird. Die beiden Attributklassen *ComRegisterFunctionAttribute* und *ComUnregisterFunctionAttribute* des Namespace *System.Runtime.InteropServices* geben jeweils die statische Methode an, die beim Registrieren bzw. beim Aufheben der Registrierung einer Assembly für die Verwendung durch COM aufgerufen werden muss:

```
[ComRegisterFunction()]
static void Reg(String regKey)
{
  Microsoft.Win32.Registry.ClassesRoot.CreateSubKey(
    regKey.Substring(18)+ "\\Implemented Categories\\" +
    "{B56A7C42-83D4-11D2-A2E9-080009B6F22B}");
}

[ComUnregisterFunction()]
static void Unreg(String regKey)
{
  Microsoft.Win32.Registry.ClassesRoot.DeleteSubKey(
    regKey.Substring(18)+ "\\Implemented Categories\\" +
    "{B56A7C42-83D4-11D2-A2E9-080009B6F22B}");
}
```

Da diese Art der Registrierung sehr aufwendig ist, stellt ESRI im Namensraum *ESRI.ArcGIS.Utility.CATIDs* Klassen für sämtliche ArcGIS-Komponentenkategorien bereit. Jede dieser Klassen repräsentiert eine Kategorie und beinhaltet Methoden zur Vereinfachung des Registrierungsprozesses. Das folgende Listing zeigt, wie eine .NET-Komponente für die MxCommands-Kategorie registriert wird:

```
#region Component Category Registration

  [ComRegisterFunction()]
  [ComVisible(false)]
  static void RegisterFunction(string regKey)
  {
    MxCommands.Register(regKey);
  }

  [ComUnregisterFunction()]
  [ComVisible(false)]
  static void UnregisterFunction(string regKey)
  {
    MxCommands.Unregister(regKey);
  }

#endregion#
```

Weiterführende Informationen zum COM-Interoperating und zur COM-Komponentenregistrierung finden sich in der ArcGIS Developer Help (DEVHELP_2006).

4.2.2 Erstellen eines Tools für ArcMap

Die *ESRI.ArcGIS.Utility*-Assembly beinhaltet zwei abstrakte Basisklassen (*BaseCommand* und *BaseTools*), die den Entwickler bei der Erstellung neuer benutzerdefinierter .NET-Erweiterungen unterstützen. Sie bieten vorgefertigte Implementierungen für alle Member der *ITool* und *ICommand*-Schnittstellen, so dass in der Klasse lediglich die Methoden und Eigenschaften überschrieben werden müssen, die das Werkzeug für sich benötigt. Die einzige Ausnahme bildet *ICommand::OnCreate*. Diese Methode wird beim Erzeugen des Tools aufgerufen und muss in jedem Fall überschrieben werden. Im Folgenden soll die Vorgehensweise bei der Erstellung neuer Werkzeug-Komponenten anhand der Measure-Funktion ausgeführt werden. Die ArcGIS Developer Help (DEVHELP_2006) enthält ausführliche Tutorials zu dieser Thematik.
Zunächst werden einem neuen Visual Studio-Projekt vom Typ „Klassenbibliothek“ die Referenzen auf die benötigten ESRI- und .NET-Assemblies hinzugefügt. Unabhängig von der Funktionalität des Werkzeugs müssen unbedingt Verweise auf die *ESRI.ArcGIS.Utility*-dll, welche die abstrakten Basisklassen *BaseCommand* und *BaseTool* sowie die Klassen der Komponentenkategorien enthält, wie auch auf die *System.Runtime.InteropServices*-dll für die Erzeugung des COM-Wrappers eingebunden werden. Der using-Block sieht schließlich folgendermaßen aus:

```
using ESRI.ArcGIS.ArcMapUI;
using ESRI.ArcGIS.Carto;
using ESRI.ArcGIS.Display;
using ESRI.ArcGIS.Framework;
using ESRI.ArcGIS.Geodatabase;
using ESRI.ArcGIS.Geometry;
using ESRI.ArcGIS.Location;
using ESRI.ArcGIS.esriSystem;
using ESRI.ArcGIS.SystemUI;
using ESRI.ArcGIS.Utility.BaseClasses;
using ESRI.ArcGIS.Utility.CATIDs;
using System.Runtime.InteropServices;
using System.Windows.Forms;
using System;
```

Im nächsten Schritt kann dem Projekt eine Bitmap-Datei hinzugefügt werden, die das Icon des Tools innerhalb der ArcMap-Anwendung repräsentiert. Das Icon kann direkt in Visual Studio graphisch bearbeitet werden (Abbildung 17):

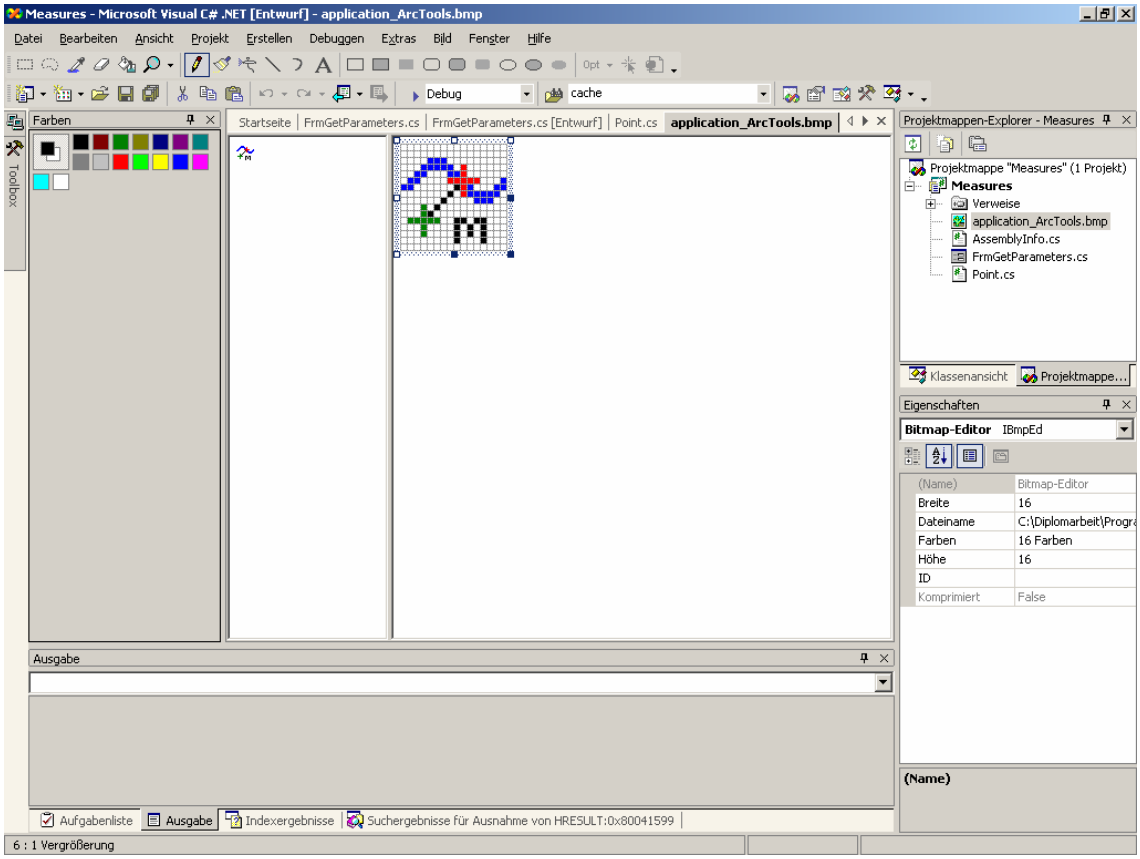

Abbildung 17: Bearbeitung des Icon

Das folgende Listing zeigt den Code des Tools vor dem Hinzufügen der eigentlichen Geo-Funktionalität. Die Klasse ist von *BaseTool* abgeleitet und erbt damit alle deren Member. Über das *base*-Schlüsselwort können im Konstruktor nun jene Member überschrieben werden, die das Tool benötigt. Die übrigen Member übernehmen die default-Werte der Basisklassenimplementierung. Dabei kann z.B festgelegt werden, in welcher Kategorie der „Customize Dialog Box“ von ArcMap das Tool bei der Kompilierung abgelegt werden soll oder welche Bitmap für das Icon verwendet wird. Wie zuvor erwähnt, muss die *onCreate*-Methode der *BaseCommand*-Klasse unbedingt überschrieben werden. Sie bewirkt, dass bei der Auswahl des Tools ein Zeiger auf die aktuelle Applikation, das aktuelle Dokument und den ActiveView erzeugt wird. Die Überschreibung von *BaseTool::OnMouseDown* ruft die gesamte Geo-Funktionaliät des Tools auf und wird immer dann ausgeführt, wenn der Anwender mit dem ausgewählten Werkzeug in die Karte klickt.

```
public sealed class Point : BaseTool
{
  private IApplication m_app;
  private IMxDocument mxDoc;
  private IActiveView m_mapView;
  private IMap m_map;
  private string CR = Environment.NewLine;

  // Der Konstruktor erzeugt das Tool und
  // deklariert die Member der BaseTool-Klasse
  public Point()
  {
    // benötigte Member des Tools festlegen
    // die restlichen Member von ICommand liefern
    // den Default-Wert
    base.m_category = "GewIS";
    base.m_caption = "Find Nearest Measure C#";
    base.m_message = "Ermittelt den zum InputPoint nächsten
      Measure";
    base.m_toolTip = "Find Nearest Measure C#";
    base.m_name = "GewIS_FindNearestMeasure C#";

    // Erzeugt eine Instanz eines Bitmap-Icons, dass in ArcMap
    // angezeigt werden kann
    string[] res =
      GetType().Assembly.GetManifestResourceNames();
    if (res.GetLength(0) > 0)
    {
      base.m_bitmap = new
        System.Drawing.Bitmap(GetType().Assembly.
        GetManifestResourceStream(res[0]));
    }
  }

  // Beim Erzeugen des Tools wird ein Pointer
  // auf die aktuelle Applikation gesetzt
  public override void OnCreate(object hook)
  {
    m_app = hook as IApplication;
    mxDoc = m_app.Document as IMxDocument;
    m_mapView = mxDoc.FocusMap as IActiveView;

  }

  // Mausklick in die Karte
  public override void OnMouseDown(int Button, int Shift,
    int X, intY)
  {
    // hier kommt die Funktionalität rein bzw.
    // von hier die Methoden aufgerufen…
    base.OnMouseDown (Button, Shift, X, Y);
  }
}
```

Um das Tool in ArcMap nutzen zu können, muss die Komponente noch, wie im vorangegangenen Kapitel beschrieben, für COM und die entsprechende COM Komponentenkategorie registriert werden. Vor der Klassendeklaration wird das GUID (Globally Uniquely Identifier)-Attribut eingefügt, durch das die Klasse eindeutig identifizierbar ist. Neue GUID's können in Visual Studio mit dem Programm GuidGen erzeugt werden, das sich im Menü „Extras" befindet. Die Klasse kann anschließend kompiliert und in der Customize Dialog Box von ArcMap zur Verwendung ausgewählt werden (Abbildung 18).

```
[ClassInterface(ClassInterfaceType.None)]
[Guid("5154158D-D42B-460b-A4E3-64B6F0F0B0B3")]

public sealed class Point : BaseTool
{
  # region Component Category Registration

  [ComRegisterFunction()]
  [ComVisible(false)]
  static void RegisterFunction(string regKey)
  {
    MxCommands.Register(regKey);
  }

  [ComUnregisterFunction()]
  [ComVisible(false)]
  static void UnregisterFunction(string regKey)
  {
    MxCommands.Unregister(regKey);
  }

  #endregion#
```

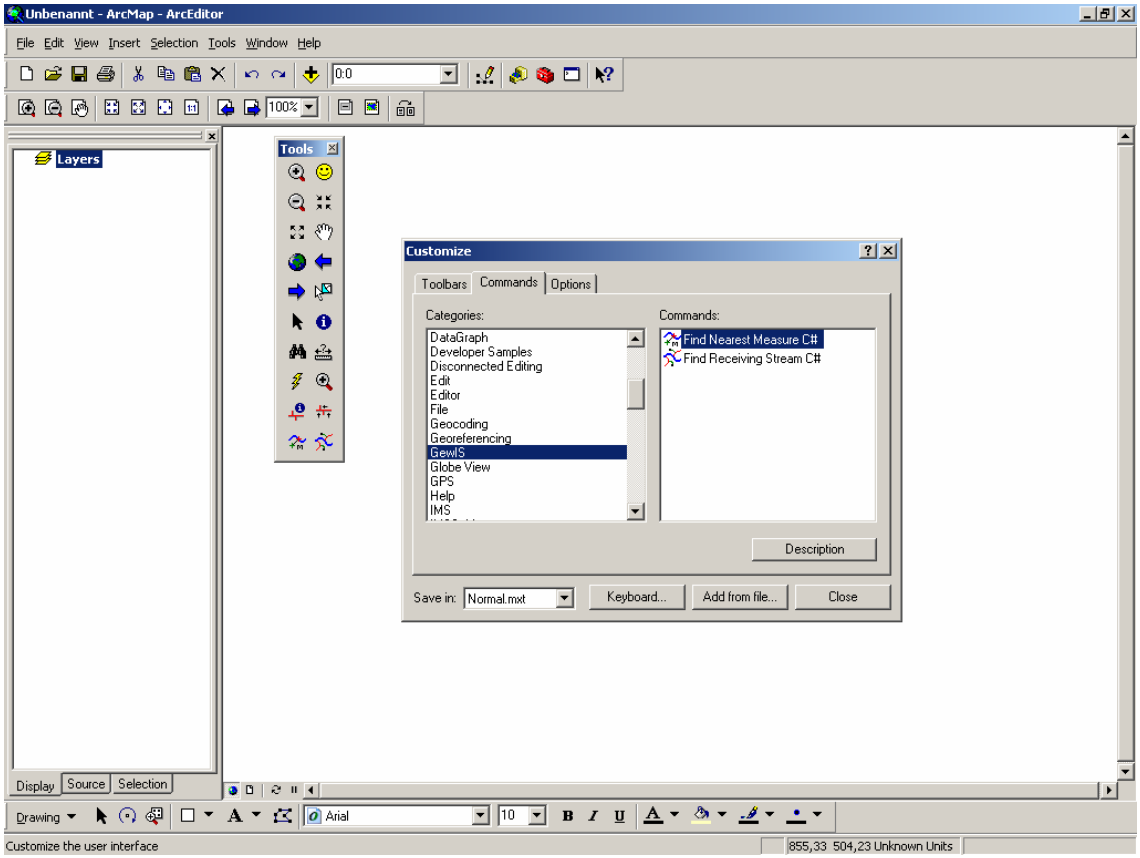

Abbildung 18: Verwendung des Tools in ArcMap

Fazit

Im Hinblick auf die Entwicklung weiterer Commands für das RipsDesktop Framework stellte sich heraus, dass einzelne Tools nicht in ein- und demselben Standard-Namespace erzeugt werden können, da sie sonst in ArcMap nicht verfügbar sind. Ebenso scheint die Interaktion zwischen ArcObjects und .NET nicht immer einwandfrei zu funktionieren. So wurde die Measure-Funktion zuerst mit Hilfe der *IRouteLocator*-Schnittstelle aus der *Location*-Library programmiert, die in .NET allerdings zu Speicherbereinigungsproblemen führte und daher nicht verwendet werden konnte. Abgesehen davon funktioniert die Programmierung neuer ArcGIS-Erweiterungen jedoch sehr gut. ESRI bietet mittlerweile auf seiner EDN-Homepage (EDN_2006) das ArcGIS Visual Studio .NET Integration Framework (Abbildung 19) an, das dem Entwickler nach der Installation zahlreiche neue Vorlagen bereitstellt, auf deren Basis spezielle ArcGIS-Klassen erzeugt werden können. Über einen Assistenten können verschiedene Klassenoptionen, die Art der Komponente und die Kategorie, in der sie

zu registrieren ist, gewählt werden. Der Wizard übernimmt anschließend die automatische Generierung des Codes und fügt der Klasse sämtliche Attribute zu, die für die Kompilierung benötigt werden.

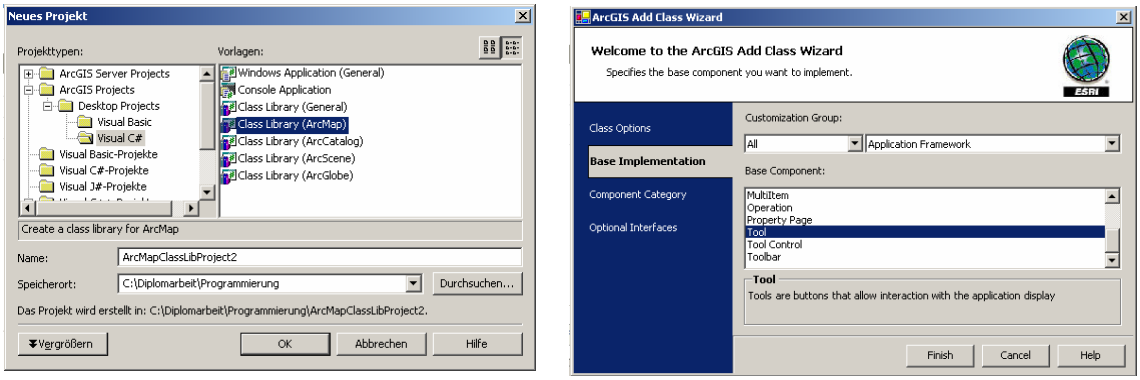

Abbildung 19: ArcGIS Visual Studio .NET Integration Framework

4.3 GIS Web Services mit dem ArcGIS Server

4.3.1 Einrichten des GIS Servers

Um Web-Applikationen und Web Services zu entwickeln, muss das Application Developer Framework (ADF) installiert sein. Für die Web Services sind die ASP.NET WebControls (Server Controls) relevant. Dieser Namespace bietet Objekte und Methoden, um sich mit dem GIS Server verbinden und dessen Funktionalität nutzen zu können.

Nach der erfolgreichen Installation des ArcGIS Server und des Frameworks kann der GIS Server konfiguriert werden. Die Post-Installation umfasst die Einrichtung der Accounts für den Server Object Manager und die Server Object Container, die Einrichtung der Nutzergruppen (Administratoren, Nutzer) sowie die Vergabe von Zugriffsrechten für alle relevanten Verzeichnisse (data directory, output directory und virtual directory). Detaillierte Informationen zur Einrichtung können dem ArcGIS Server Administrator and Developer Guide (ESRI, 2004) sowie den ArcGIS Server Schulungsunterlagen entnommen werden.

Über ArcCatalog haben Mitglieder der Nutzergruppen nun die Möglichkeit, eine Verbindung mit dem Server aufzubauen und neue Server Objects anzulegen. Dabei

müssen die in Kapitel 3.3.2 erläuterten Parameter berücksichtigt werden. ArcCatalog bietet zudem eine Reihe verschiedener Möglichkeiten, die Nutzung der Objekte durch Client-Applikationen zu analysieren.
Für die testweise Entwicklung der Web Services wurde zunächst ein lokaler GIS Server eingerichtet, auf dem das kilometrierte AWGN (Amtliches wasserwirtschaftliches Gewässernetz) als MapServer Object abgelegt wurde. Später wurde ein Produktionsserver im Netzwerk der LUBW („gissrv1“) verwendet.

4.3.2 XML Web Services mit Visual Studio .NET 2003

Visual Studio stellt für die Entwicklung von Web Services die ASP.NET-Webdienst-Vorlage bereit. Standardmäßig werden beim Anlegen des Projektes vier Dateien erzeugt:

- *AssemblyInfo.cs (Metadaten auf die Assemblies des Projektes)*
- *Global.asax (ASP.NET-Anwendungsdatei)*
- *Service1.asmx (Hauptlogik des Web Service)*
- *Web.config (Konfigurationseinstellungen für den Dienst)*

Das Template importiert neben *System* und *System.Data* automatisch die *System.Web*-, *System.Web.Services*- und die *System.Xml*-Bibliotheken.

System.Web

Dieser Namensraum bietet Klassen und Schnittstellen für die Kommunikation zwischen dem Browser und dem Server. Dazu gehören unter anderem die *HttpRequest* und *HttpResponse*-Klassen für die Verarbeitung der Http-Anforderungen und –Ausgaben.

System.Web.Services

Beinhaltet Klassen, die das Erstellen von XML Web Services unter Verwendung von ASP.NET ermöglichen. Zwei Klassen sind von besonderer Bedeutung: Die *WebService*-Klasse definiert die Basisklasse für die Webdienste und ermöglicht den Zugriff auf gemeinsam genutzte ASP.NET-Objekte wie den Application- oder Session

State. Die zweite ist die *WebMethodAttribute*-Klasse, welche dafür sorgt, dass der Web Service von Remote Web Clients aufgerufen werden kann. Diese Klasse bietet unter anderem folgende öffentliche Eigenschaften:

- *Namespace:*

  Sie gibt den Namensraum an, über den ein Web Service identifiziert werden kann.

- *MessageName:*

  Mit dieser Eigenschaft kann ein Alias für einen Methodennamen vergeben werden. Das ist notwendig, um polymorphe Methoden des Web Service eindeutig zu identifzieren.

- *Description:*

  Die Description-Eigenschaft liefert eine Beschreibung des Web Service.

System.Xml

Der Namespace stellt auf verschiedenen Standards aufbauende Unterstützung bei der XML-Verarbeitung bereit.

4.3.3 Application Web Services mit ArcGIS Server

Im vorigen Kapitel wurde erläutert, wie mit Hilfe des Programmiersystems ASP.NET XML Web Services erstellt werden können. Durch ArcGIS Server ist es möglich, GIS-Funktionalitäten in diese Dienste zu integrieren. Dabei werden grundsätzlich zwei Arten von Web Services unterschieden: Application Web Services und ArcGIS Server Web Services. Bei den letzteren handelt es sich um Server Objects, die über ihre SOAP-Schnittstelle direkt als Dienst auf dem GIS Server angeboten werden können (vgl. Kapitel 3.3.2).
Application Web Services werden dagegen wie die gewöhnlichen XML Web Services auf Basis des Web Service Framework ASP.NET erstellt. Dabei ist zu beachten, dass

der Webdienst keine direkten ArcObjects als Parameter erhält oder als Rückgabewert liefert, sondern nur die nativen Datentypen, die vom Framework unterstützt werden (ESRI, 2004).
Prinzipiell unterscheidet sich die ArcObjects-Programmierung über die Server API nicht wesentlich von der Desktop-Version. Es sind lediglich folgende Details zu beachten:

- *Wie erhält man eine Verbindung zum Server?*
- *Wie erhält man Zugriff auf die Server Objects?*
- *Wie erzeugt man neue Objekte auf dem Server?*

Abbildung 20 zeigt die Klassen und Schnittstellen, die den Zugriff auf die MapServer- und GeocodeServer Objects ermöglichen.

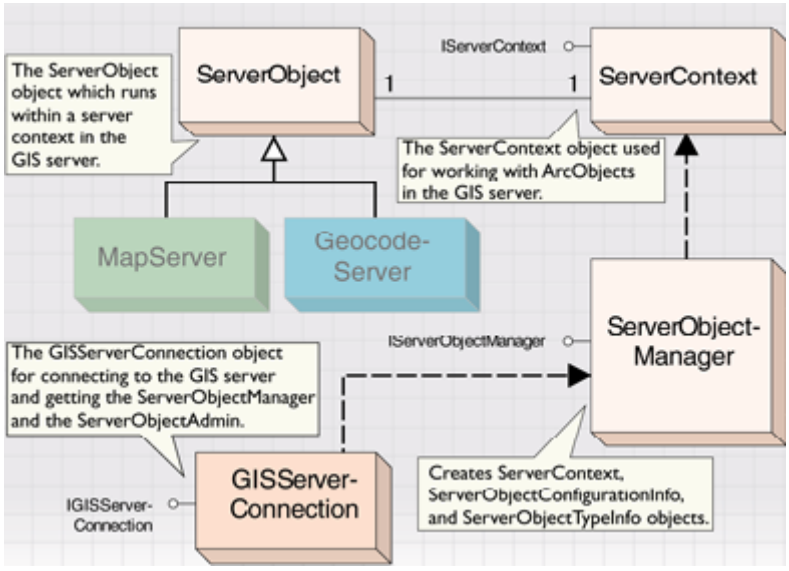

Abbildung 20: Server Consumer Objects

4.3.3.1 Verbindung zum GIS Server

Um die ArcObjects des GIS Servers nutzen zu können, muss zunächst über das *ServerConnection*-Objekt (Abbildung 21) eine Referenz zum Server Object Manager hergestellt werden. Dabei verwendet jede Laufzeitumgebung ihr eigenes Connection-Objekt. Das Objekt für .NET und Java befindet sich in der

ESRI.ArcGIS.Server.WebControls-Assembly, so dass für .NET-Projekte diese Bibliothek eingebunden werden muss. Im *ServerConnection*-Interface wird die Methode *Connect* definiert, die eine Verbindung zum zuvor festgelegten Host aufbaut.

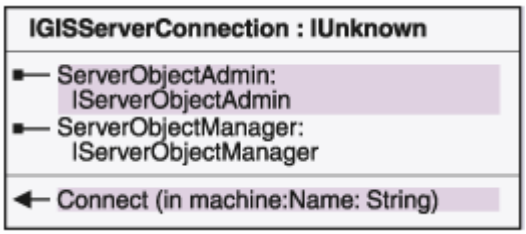

Abbildung 21: IGISServerConnection-Interface

Bei der Verwendung von ASP.NET muss sich der Nutzer beim GIS Server authentifizieren, da eine Anmeldung sonst nicht erfolgen kann. Voraussetzung hierfür ist, dass der User-Account der agsusers- oder agsadmin-Gruppe des GIS Servers angehört und die entsprechenden Zugriffsrechte besitzt. Für die Authentifizierung (Impersonation) muss die Web.Config-Datei um den *identity*-Eintrag ergänzt werden. Die Authentifizierungsparameter können entweder, wie im folgenden Code-Beispiel, in der Web.Config-Datei direkt, oder in der Verzeichnissicherheit des virtuellen Verzeichnisses auf dem Web Server eingetragen werden:

```
<?xml version="1.0" encoding="utf-8" ?>
  <configuration>
    <system.web>
      <identity impersonate="true" userName="LFUBK\53_D_SH"
        password="anfangen1" />
…
    </system.web>
  </configuration>
```

4.3.3.2 Server Objects – Zugriff und Erzeugung

Sobald die Verbindung zum GIS Server besteht, kann mit den Server Objects gearbeitet werden. Dazu instanziiert die *ServerObjectManager*-Klasse über die Methode *CreateServerContext* ein Objekt vom Typ *IServerContext* (Abbildung 22). Dieses besitzt wiederum die Eigenschaft *ServerObject*, welche das grobkörnige MapServer- bzw. GeocodeServer Object des aktuellen Kontexts enthält. Im nächsten

Listing ist zu sehen, wie mit der Methode *get_Map* der *IMapServerObjects*-Schnittstelle auf das Map-Objekt zugegriffen wird. Mit diesem Objekt kann nun gearbeitet werden, als würde es sich um eine generische ArcObjects-Komponente handeln. Der einzige Unterschied zur Desktop-Programmierung besteht in der Erzeugung neuer Objekte. Diese erfolgt nicht mit dem *new*-Operator, sondern mit der *CreateObject*-Methode der *IServerContext*-Schnittstelle.

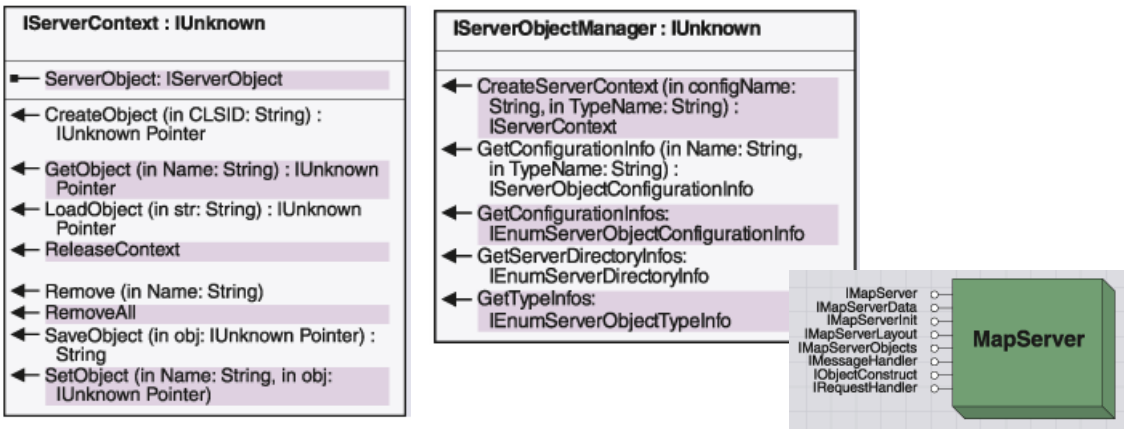

Abbildung 22: IServerObjectManager, IServerContext und MapServer

Die Methode *CreateServerContext* bietet auch die Möglichkeit, einen leeren ServerContext zu erzeugen, in dem ArcObjects auf dem Server „on the fly“ generiert werden können. Das ist sinnvoll, wenn Objekte für die eigene Applikation erzeugt werden sollen, die aber keine vorkonfigurierten Server Objects benötigen. Leere ServerContexts sind standardmäßig nonpooled und haben eine hohe Isolation.

Wird ein ServerContext nicht mehr benötigt, muss er von der Anwendung durch den Aufruf der *ReleaseContext*-Methode wieder freigegeben werden, um die Objekte für andere Application Sessions verfügbar zu machen. Andernfalls bleibt das Objekt solange belegt, bis der Garbage Collector der Laufzeitumgebung die Freigabe übernimmt. Der folgende Code zeigt den Aufbau der Web-Methode *FindNearest()* des Measure-Web Service vor der Implementierung der Geo-Funktionalität:

```
// Web-Methode
[WebMethod(MessageName="FindNearest(x,y,searchRadius)",
Description="Bestimmt den zum InputPoint nächstliegenden
Measure")]
public MeasureVal[] FindNearest(double p_dSearchRadius,
  double p_dX, double p_dY)
{
  using (WebObject webObj = new WebObject())
  {
    // Verbindung zum GIS Server
    ESRI.ArcGIS.Server.WebControls.ServerConnection connection=
      new ESRI.ArcGIS.Server.WebControls.ServerConnection();
    connection.Host = "gissrv1";
    connection.Connect();

    // Server Context aufbauen
    IServerObjectManager som = connection.ServerObjectManager;
    IServerContext ctx = som.CreateServerContext("fluss10_km",
      "MapServer");

    try
    {
      // Zugriff auf das Map-Objekt
      IMapServer mapSrv = ctx.ServerObject as IMapServer;
      IMapServerObjects mapSrvObjects =
        (IMapServerObjects) mapSrv;
      IMap map = mapSrvObjects.get_Map(mapSrv.DefaultMapName);

      // Verarbeitung, von hier an wie mit gewöhnlichen
      // ArcObjects…

      // return-value
    }
    catch
    {
      throw;
    }
    finally
    {
      // ServerContext freigeben
      ctx.ReleaseContext();
    }
    return null;
  }
}
```

Das Helfer-Objekt *WebObject* aus dem WebControls-Namespace sorgt dafür, dass sämtliche Referenzen auf COM-Objekte (z.B. auf einen FeatureCursor) gelöscht werden, sobald dessen Gültigkeitsbereich verlassen wird. Dazu werden die Objekte mit der Methode *ManageLifetime* einer Liste der von WebObject zu verwaltenden Komponenten hinzugefügt (ESRI, 2004).

4.3.3.3 Testen eines Web Service

Das ASP.NET-System bietet Hilfsseiten zum einfachen Austesten von XML Web Services mit dem Browser (Abbildung 23). Nach Starten des Visual Studio-Projektes besteht die Möglichkeit, die Web-Methoden über eine vorgefertigte Benutzerschnittstelle aufzurufen. Neben dieser Maske enthalten die Seiten allgemeine Informationen zum Webdienst, den bereitgestellten Methoden sowie Beispiele für SOAP-Anfragen und –Antworten. Durch Drücken der Schaltfläche „Aufrufen“ wird die Web-Methode über das HTTP-POST Protokoll gestartet. Die zurückgelieferten Daten werden in Form eines XML-Dokuments in einem neuen Browser angezeigt (Abbildung 24).

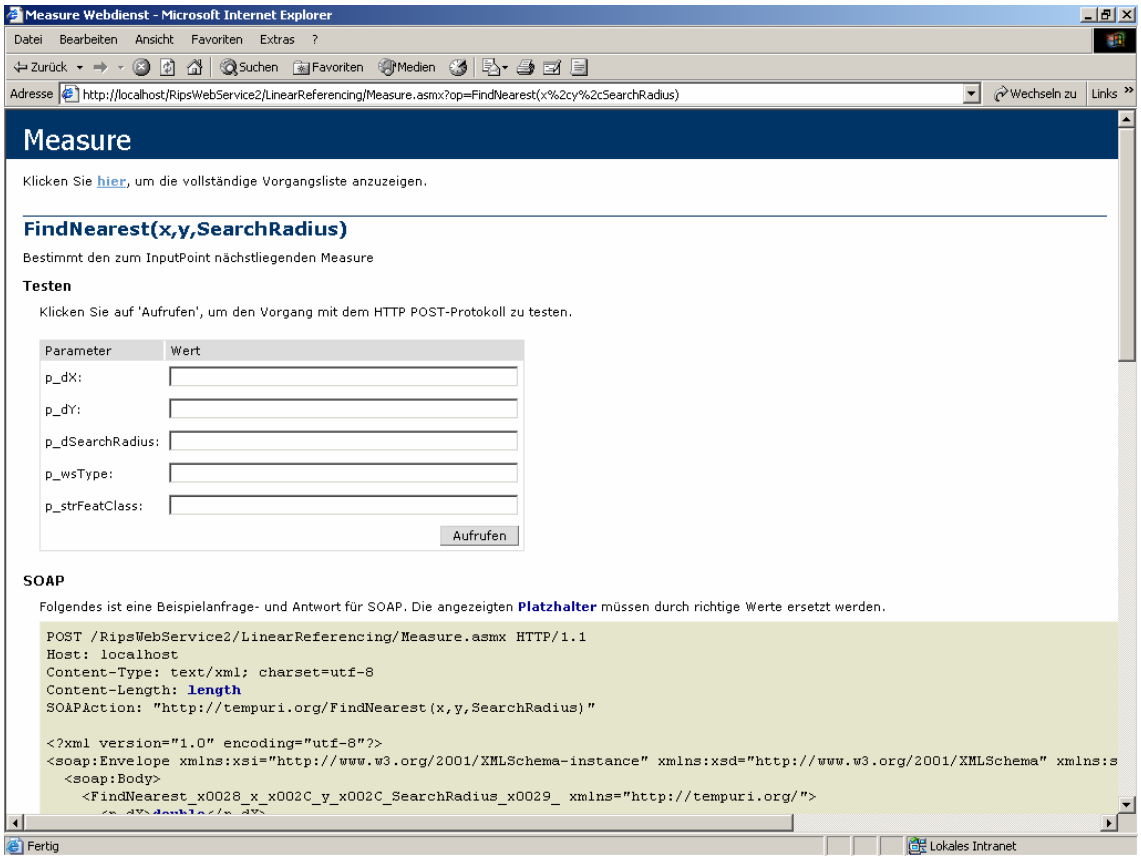

Abbildung 23: Maske zum Testen einer Web-Methode

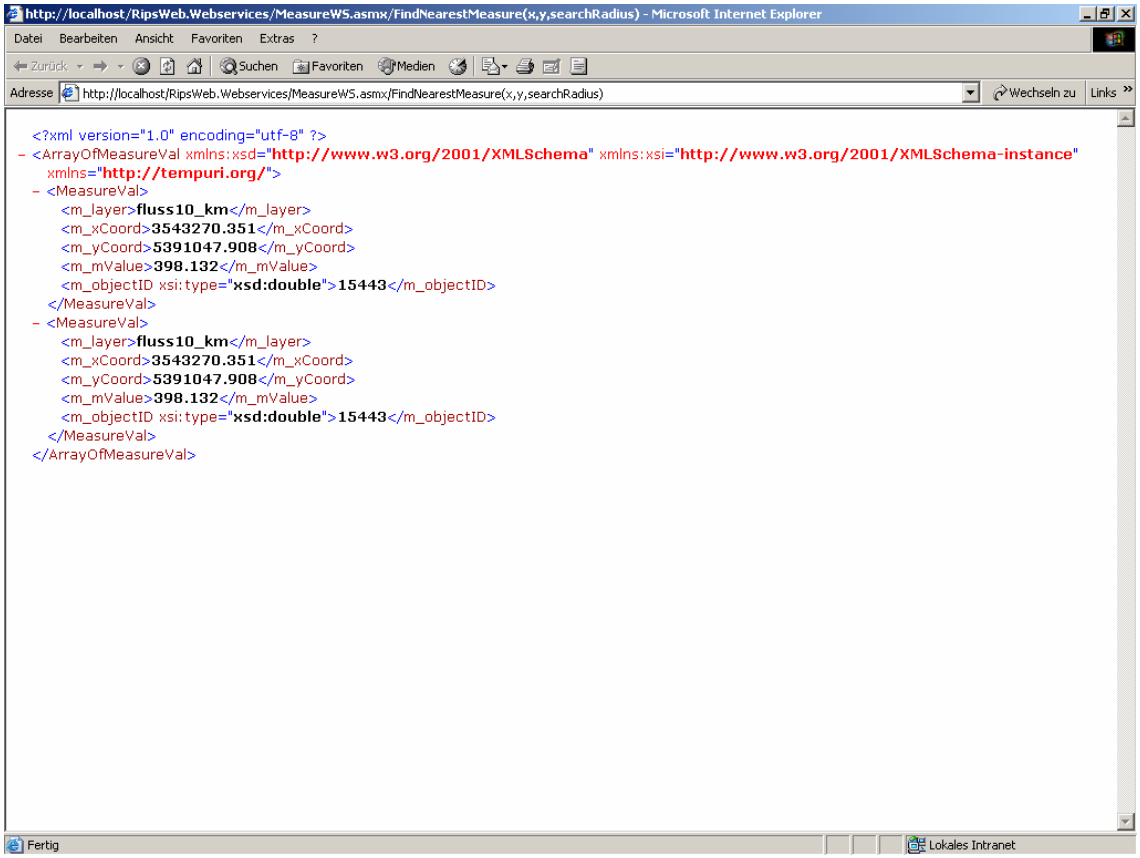

Abbildung 24: Rückgabewerte der Web-Methode

4.4 Client-Applikationen

XML Web Services dienen in erster Linie dem Datenaustausch zwischen verschiedenen Anwendungen. Bei den Konsumenten kann es sich beispielsweise um andere XML-Dienste, Windows/WebForm-Anwendungen oder auch Server Prozesse handeln (Freeman&Jones, 2003). Dabei spielt es keine Rolle, in welcher Programmiersprache und auf welcher Plattform der Web Service erstellt wurde. Der Konsument muss lediglich die Struktur der Nachricht, die der Web Service voraussetzt, sowie das verwendete Kommunikationsprotokoll kennen. Diese Informationen können dem WSDL-Dokument entnommen werden. Im Rahmen dieses Kapitels werden drei Test-Clients für .NET, HTML/JavaScript und Microsoft Office erstellt.

4.4.1 .NET-WebForm

Die Erstellung von .NET-Clients in Visual Studio ist durch die Verwendung von Proxyklassen relativ einfach zu bewerkstelligen. Proxyklassen sind gewöhnliche Klassen, welche die vom Web Service bereitgestellte Funktionalität widerspiegeln und dem Entwickler die Generierung der SOAP-Anfragen abnehmen (Freeman&Jones, 2003). Abbildung 25 verdeutlicht die Kommunikationswege zwischen dem Client und dem XML Web Service. Um die Funktionen des Dienstes zu nutzen, ruft der Konsument die Methoden der Proxyklasse auf. Diese regelt die Interaktion mit dem Web Service und liefert das Ergebnis an den Client zurück, wobei der Aufruf der Methoden innerhalb der Client-Anwendung so erfolgt, als würde es sich um lokale Elemente handeln.

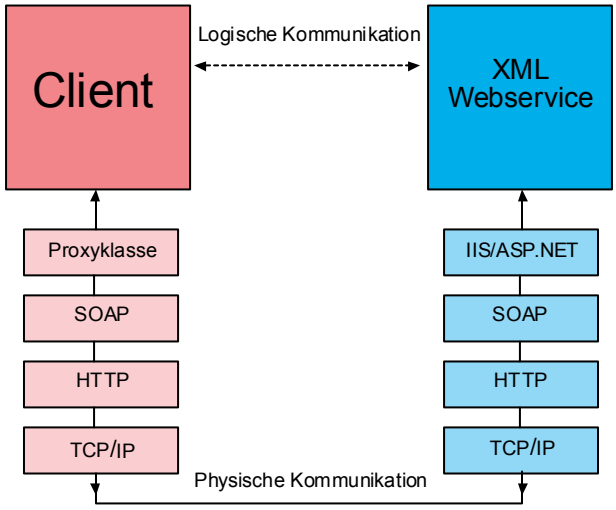

Abbildung 25: Kommunikation über eine Proxyklasse

.NET-Clients sind Web-Applikationen, die auf dem Web Server ausgeführt werden und auf die über einen Browser zugegriffen wird. Zur Erstellung in Visual Studio wird die ASP.NET-Webanwendungsvorlage ausgewählt. Im ersten Schritt wird im Form Designer eine Web Form erzeugt, wie sie in Abbildung 27 zu sehen ist. Anschließend kann die Proxyklasse erstellt werden. Visual Studio übernimmt diese Aufgabe automatisch, in dem ein Webverweis auf die asmx-Datei des Service-Projektes eingefügt wird (Abbildung 26). In den Eigenschaften dieses Verweises kann das URL-Verhalten entweder auf „static“ oder auf „dynamisch“ gesetzt werden. Wird „static“ verwendet, so fügt Visual Studio die URL des Web Service hartverdrahtet in die

Proxyklasse ein, wohingegen sie bei dynamischem Verhalten in die Konfigurationsdatei (im Element *<appSettings>*) eingetragen und dann zur Laufzeit eingelesen wird. Das hat den Vorteil, dass die Adresse des Web Service geändert werden kann, ohne dass die Proxyklasse angepasst werden muss.

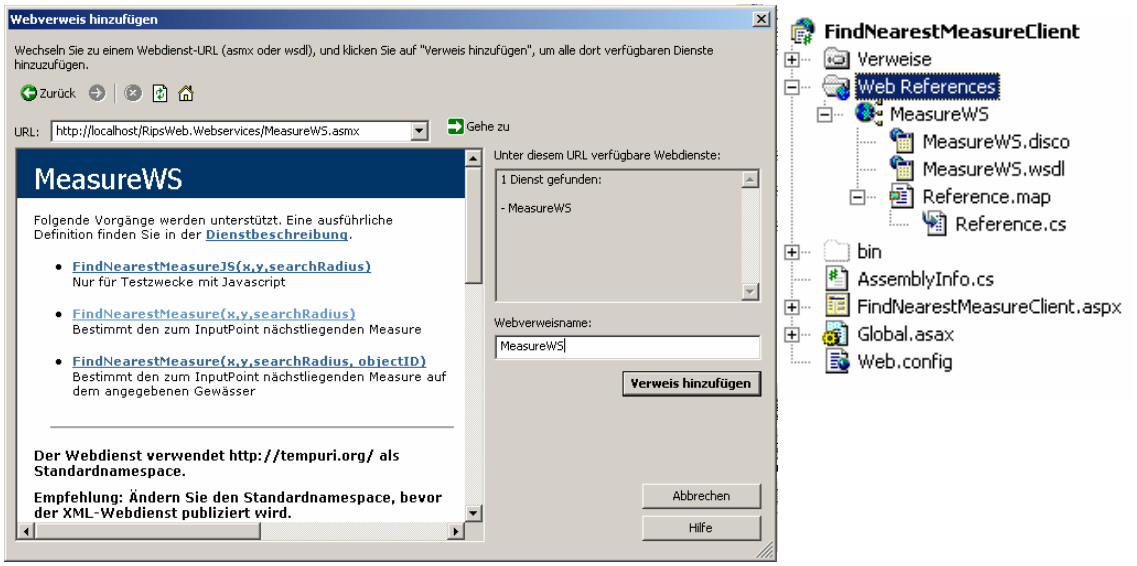

Abbildung 26: Einfügen einer Web Reference

Um den Proxy zu aktivieren, muss dessen Namensraum in die Klasse importiert werden. Damit sind sämtliche Methoden des Web Service verfügbar:

```
// using Statements
// …
// importiert den Namensraum des Webdienstproxy
using FindNearestMeasureClient.MeasureWS;

namespace FindNearestMeasureClient
{
  public class FindNearestMeasureClient : System.Web.UI.Page
  {
    // …
  }
}
```

Die folgende Methode wird aufgerufen, sobald der Start-Button auf der HTML-Seite gedrückt wird. Zunächst erfolgt die Datentyp-Konvertierung aus den Textfeldern. Dann wird ein Objekt der Proxyklasse *MeasureWS* erzeugt, über das die Methoden des Web Service aufgerufen werden können, als seien sie lokal vorhanden. Die SOAP-

Verarbeitung erfolgt dabei völlig automatisch. Das vom Web Service zurückgelieferte Array kann schließlich in einer einfachen HTML-Tabelle ausgegeben werden (Abbildung 27).

```
private void btnStart_Click(object sender, System.EventArgs e)
{
  // Konvertierung der Parameter
  double l_dX = Double.Parse(txtBoxX.Text);
  double l_dY = Double.Parse(txtBoxY.Text);
  double l_dRadius = Double.Parse(txtBoxRadius.Text);

  try
  {
    // Aufruf der WebService-Methode über die Proxyklasse
    MeasureWS.MeasureWS mws = new MeasureWS.MeasureWS();
    object[] l_objs = mws.FindNearestMeasure(l_dRadius, l_dX,
      l_dY);

    // Ausgabe in einer Tabelle…
  }
  catch
  {
    throw;
  }
}
```

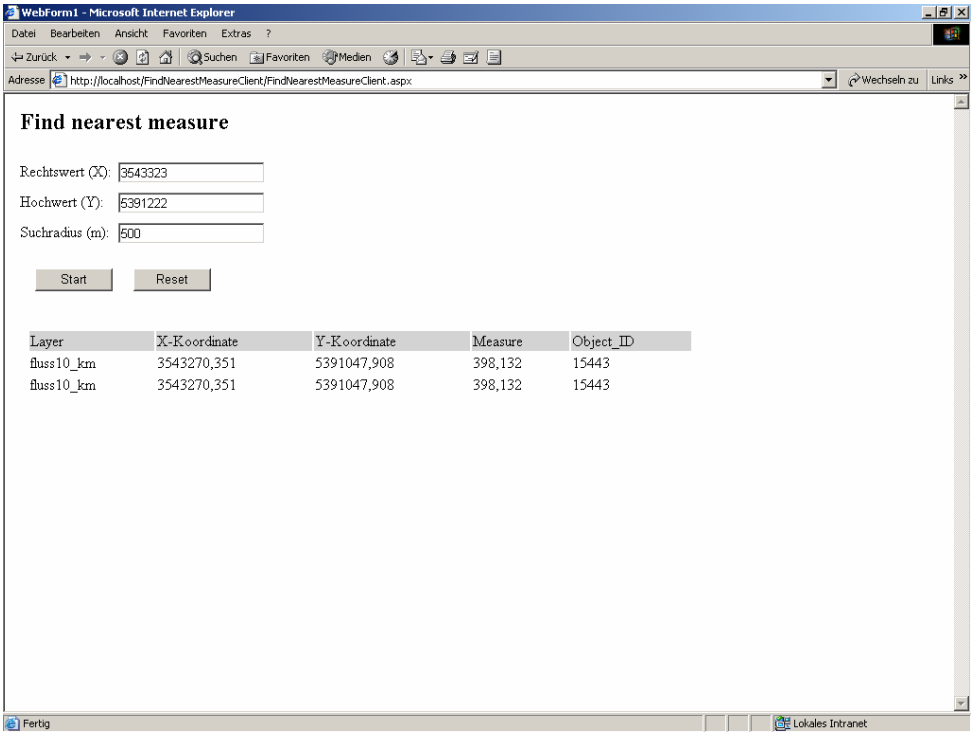

Abbildung 27: .NET WebForm Client

4.4.2 MS-Office

Um Web Services in Microsoft Office-Anwendungen einbinden zu können, muss das MS Office Web Services Toolkit installiert werden. Die Software ist sowohl für Office XP als auch für Office 2003 erhältlich und kann im Download-Bereich der Microsoft-Homepage heruntergeladen werden. Sie ermöglicht die schnelle Integration von XML-Diensten in VBA-Programme. Liefert der Web Service komplexe Datentypen zurück, so ist unbedingt darauf zu achten, dass mindestens die Version 2.0 verwendet wird.
Das Web Services Toolkit stellt eine Schnittstelle zur Verfügung, um über UDDI (vgl. Kapitel 3.1.3) nach registrierten Web Services zu suchen oder diese direkt über ein WSDL-File in ein Projekt einzubinden (Abbildung 28). Nach der Installation kann das Werkzeug im VBA-Editor der Office-Anwendung gestartet werden. Durch die Eingabe einer URL erhält man Zugriff auf den gewünschten Web Service und kann diesen vor der Einbindung über die ASP.NET-Testseiten überprüfen. Sobald der Verweis dem Projekt hinzugefügt wird, erzeugt der VBA-Editor automatisch die benötigten Klassenmodule (Abbildung 29).

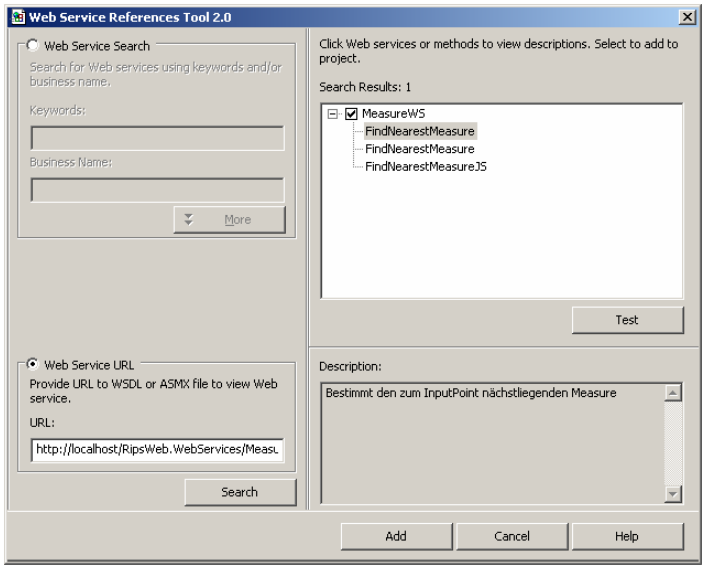

Abbildung 28: Das Web Service Reference Tool für MS-Office

Im Rahmen dieser Untersuchung wurde ein kleines Excel-Makro erstellt, das die gleiche Funktionalität bietet, wie der .NET WebForm Client. In einer Tabelle können Hoch- und Rechtswert sowie der Suchradius eingegeben werden. Nach Drücken des

GetMeasure(s)-Buttons wird die Web-Methode aufgerufen und die Tabelle mit den zurückgelieferten Daten ergänzt (Abbildung 29).

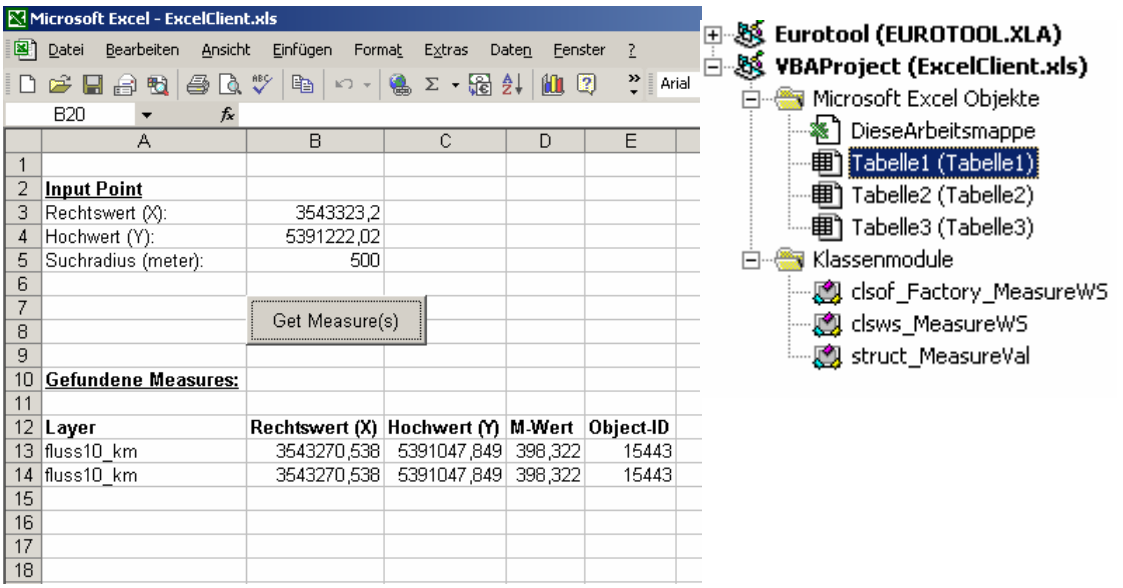

Abbildung 29: Microsoft Excel Client und die Klassenmodule

Nachfolgend ist der VBA-Code des Makros zu sehen. Auch hier wird, wie im .NET-Client, ein Objekt einer Proxyklasse erzeugt, über das die Methoden des Web Service aufgerufen werden können.

```
Private Sub CommandButton2_Click()
   'Objekt der WebService-Proxyklasse
    Dim ws As New clsws_MeasureWS

    'Parameter, die an den Webservice übergeben werden
    Dim pXCoord As Double
    Dim pYCoord As Double
    Dim pSearchRadius As Double

    'Array, welches die Ergebnisse des Webservices aufnimmt
    Dim result() As Object

    'Abfrage der Parameterwerte aus der Anwendung
    pXCoord = Range("B3").Text
    pYCoord = Range("B4").Text
    pSearchRadius = Range("B5").Text

    'Aufruf des Webservice
    result = ws.wsm_FindNearestMeasure(pSearchRadius, pXCoord,
```

```
          pYCoord)

        'Ausgabe des Arrays
        Dim row As Integer
        row = 13
        For i = 0 To UBound(result)

          'Objekt der struct_MeasureVal erzeugen
          Dim msv As struct_MeasureVal
          Set msv = result(i)

          'Werte in der Tabelle ausgeben
          Tabelle1.Cells(row, 1).Value = msv.m_layer
          Tabelle1.Cells(row, 2).Value = msv.m_xCoord
          Tabelle1.Cells(row, 3).Value = msv.m_yCoord
          Tabelle1.Cells(row, 4).Value = msv.m_mValue
          Tabelle1.Cells(row, 5).Value = msv.m_objectID

          row = row + 1
        Next
    End Sub
```

Bei der Ausführung des Makros gab es zunächst Probleme mit der Typverarbeitung. Der Web Service lieferte die Objekt-ID des Gewässers als Wert vom Typ *object* zurück, was zu folgender Fehlermeldung führte:

> *„SoapMapper: Restoring data into SoapMapper MeasureVal failed".*

Das Problem konnte behoben werden, in dem der automatisch generierten *struct_MeasureVal*-Klasse des Excel-Makros der Typ der Klassenvariablen *m_objectID* von *MSXML2.IXMLDOMNode* zu *double* geändert wurde.

4.4.3 HTML/JavaScript

Im RIPS-Umfeld sind mehrere Viewer im Einsatz, die auf HTML und JavaScript basierend, unterschiedliche Dienste anbieten. Es gibt verschiedene Möglichkeiten, über JavaScript auf Web Services zuzugreifen (Wenz, 2005). Als problematisch erweist sich dabei jedoch die Abhängigkeit vom genutzten Browser, die sich aus den unterschiedlichen Vorgehensweisen der einzelnen Hersteller ergibt. So führte beispielsweise Microsoft mit der Version 5 des Internet Explorer so genannte

„Behaviours“ ein, die in Form von HTML-Anweisungen den Zugriff auf Web Services über den Browser ermöglichen.

4.4.3.1 Microsoft Internet Explorer

Voraussetzung für die Einbindung von XML-Diensten ist die Datei „Webservice.htc“, welche im Download-Bereich von Microsoft bezogen werden kann. Diese Datei besteht aus ca. 2300 Zeilen JavaScript-Code, der sämtliche Funktionen für die Abwicklung der Kommunikation mit dem Web Service bereitstellt. Als Basis dient ein einfaches HTML-Dokument, in das zunächst das Web Service Behaviour eingebaut wird:

```
<body>
  <div id="FindNearestMeasureService"
       style="behavior:url(webservice.htc)">
  </div>
<body>
```

Dann kann über das Behaviour und die Methode *useService* eine Verbindung zum Web Service aufgebaut werden. Dazu müssen zwei Parameter übergeben werden: Die WSDL-Beschreibung und die Bezeichnung, unter der auf den Web Service zugegriffen werden soll. Anschließend wird mit *callService* die gewünschte Web-Methode aufgerufen:

```
//Das Behevior hat die id „FindNearestMeasureService“
FindNearestMeasureService.useService("http://localhost/RipsWeb.
Webservices/MeasureWS.asmx?WSDL", "Measures");

FindNearestMeasureService.Measures.callService(wsResult,
"FindNearestMeasureJS", radius, xCoord, yCoord);
```

In diesem Fall erfolgt der Aufruf des Web Service asynchron, d.h. der Aufruf und die Auswertung der Ergebnisse werden in zwei Schritten ausgeführt. Die *callService*-Methode erhält als Parameter unter anderem den Namen der Methode, die das Ergebnis verarbeiten soll, sobald die Antwort des Service eingetroffen ist:

```
function wsResult(result) {
  if (result.error) {
    ausgabe.innerText="Fehler!";
  } else {
    createTable(result);
  }
}
```

Das zurückgelieferte Objekt besitzt die Eigenschaften *error* und *value*. *Error* beinhaltet einen Wert vom Typ „Boolean“, der angibt, ob bei der Kommunikation mit dem Web Service ein Fehler aufgetreten ist. „*Value“* enthält den eigentlichen Ergebniswert.

4.4.3.2 Mozilla

Auch Mozilla unterstützt ab Version 0.9.9. den Zugriff auf Webdienste mittels JavaScript. Allerdings sind die SOAP-Erweiterungen nicht ausgereift und weisen noch einige Fehler auf (Wenz, 2005). So funktionierte der Aufruf des Web Service im Rahmen dieser Untersuchung leider nicht einwandfrei. Dennoch soll kurz die prinzipielle Vorgehensweise erklärt werden, da diese Schwierigkeiten in zukünftigen Versionen behoben werden sollen.
Die Mozilla API stellt für den Aufruf von Web Services das *SOAPCall*-Objekt bereit. Die *transportURI*-Eigenschaft dieses Objektes enthält die Adresse des Web Service, der *actionURI* die Methode, die aufgerufen werden soll:

```
var s = new SOAPCall();
s.transportURI =
  "http://localhost/RipsWeb.Webservices/MeasureWS.asmx";
s.actionURI = "http://tempuri.org/FindNearestMeasure";
```

Erwartet der Web Service Parameter, so müssen *SOAPParameter*-Objekte erzeugt werden:

```
var xCoord = new SOAPParameter();
var yCoord = new SOAPParameter();
var radius = new SOAPParameter();
xCoord.value = document.getElementById('xWert').value;
yCoord.value = document.getElementById('yWert').value;
radius.value = document.getElementById('radius').value;
```

Da Mozilla derzeit noch einen anderen Namespace für das XML-Schema verwendet als .NET, muss dieser eventuell eingebunden werden. Dieses Problem soll allerdings in zukünftigen Versionen behoben sein. Anschließend kann aus den Angaben der SOAP-Aufruf generiert und abgeschickt werden:

```
s.encode(0, "FindNearestMeasure", "http://tempuri.org/", 0,
null, 3, new Array(radius, xCoord, yCoord));
var aufruf = s.asyncInvoke(ergebnis);
```

Der Parameter von *asyncInvoke* gibt an, welche Funktion beim Eintreffen des Ergebnisses aufgerufen wird.

4.4.3.3 Zugriff mit XMLHTTP/XMLHttpRequest

Neben diesen Browser-spezifischen Lösungen besteht in der Verwendung der noch jungen XMLHTTP-Technik (MSDN_2006), die ursprünglich von Microsoft erfunden wurde und seit der Version 5 im Internet Explorer als ActiveX-Objekt integriert ist, eine weitere Möglichkeit zur Einbindung von Web Services. XMLHTTP umfasst eine API zur Datenübertragung über das HTTP-Protokoll und kann zurückgelieferte XML-Antworten entweder als Plaintext oder als DOM-Baumstruktur (Document Object Model) wiedergeben. DOM eignet sich insbesondere zur Kommunikation mit Web Services. Mittels JavaScript können die einzelnen Knoten des XML-Dokuments in JavaScript-Objekte umgewandelt und weiterverarbeitet werden. Der Vorteil dieser Technik gegenüber den bisher vorgestellten liegt in der breiten Akzeptanz durch die Browserhersteller. Neben dem Internet Explorer unterstützen mittlerweile auch die neueren Versionen von Mozilla, Safari, Opera und Gecko diese API. Außerdem ist der Umgang mit komplexen Datentypen relativ einfach zu handhaben. Als problematisch erweist sich jedoch die Tatsache, dass ältere Browser die Technik noch nicht unterstützen und somit Alternativen bereitgestellt werden müssen.
Ausgangsbasis für den Test-Client ist ein HTML-Dokument, das eine Maske zur Eingabe der Parameter enthält (Abbildung 30). Sobald der Start-Button gedrückt wird, erfolgt der Aufruf der JavaScript-Methode *findNearestMeasure().* Hier werden zunächst die Parameter ausgelesen und mit der Funktion *getHTTPObject* das XMLHttp-Objekt

erzeugt. Da der Internet Explorer eine andere Syntax als Mozilla oder Safari verwendet, wird diese Methode Browser-übergreifend implementiert:

```
//erzeugt das XMLHTTPRequest-Objekt abhängig vom genutzten
Browser
http = getHTTPObject();

function getHTTPObject() {

  var xmlhttp;
  /*@cc_on
  @if (@_jscript_version >= 5)
    try {
      xmlhttp = new ActiveXObject("Msxml2.XMLHTTP");
    } catch (e) {
      try {
        xmlhttp = new ActiveXObject("Microsoft.XMLHTTP");
      } catch (E) {
        xmlhttp = false;
      }
    }
  @else xmlhttp = false;
  @end @*/
  if (!xmlhttp && typeof XMLHttpRequest != 'undefined') {
    try {
      xmlhttp = new XMLHttpRequest();
    } catch (e) {
      xmlhttp = false;
    }
  }
  return xmlhttp;
}
```

Anschließend erfolgt die Generierung der SOAP-Anfrage an den Server. Mit der *open*-Methode wird der XMLHttpRequest initialisiert, in dem die Art der Anfrage und die URL des Web Service definiert werden. Außerdem wird mit einem Boolean-Wert festgelegt, ob der Dienst synchron oder asynchron aufgerufen wird. Erfolgt der Aufruf asynchron, muss die *onreadystate*-Eigenschaft gesetzt werden, damit bei Eintreffen des Rückgabewertes die Funktion zu dessen Verarbeitung aufgerufen werden kann (hier: *wsResult*).

```
http.open("POST", "http://localhost/RipsWeb.Webservices/
  MeasureWS.asmx", true);
http.onreadystatechange = wsResult;
```

Im nächsten Schritt werden der Content-Type-Header und der SOAP-Action-Header definiert. Mit der Methode *send()* wird die SOAP-Nachricht an den Server übertragen. Alle Angaben, die für den Aufbau des SOAP-Ausdruckes benötigt werden, befinden sich im WSDL-Dokument des Web Service. Ein Beispiel für eine SOAP-Anfrage an den Service findet sich aber auch immer auf den Web Service-Testseiten (vgl. Kapitel 4.3.3.3).

```
http.setRequestHeader("Content-Type", "text/xml");
http.setRequestHeader("SOAPAction","http://tempuri.org/
                      FindNearestMeasure(x,y,searchRadius)");
http.send(" \
  <soap:Envelope xmlns:xsi='http://www.w3.org/2001/XMLSchema-
    instance' \
       xmlns:xsd='http://www.w3.org/2001/XMLSchema' \
       xmlns:soap='http://schemas.xmlsoap.org/soap/envelope/'>\
    <soap:Body> \

<FindNearestMeasure_x0028_x_x002C_y_x002C_searchRadius_x0029_
       xmlns='http://tempuri.org/'> \
         <p_dSearchRadius>" + radius + "</p_dSearchRadius> \
         <p_X>" + xCoord+ "</p_X> \
         <p_Y>" + yCoord + "</p_Y> \

</FindNearestMeasure_x0028_x_x002C_y_x002C_searchRadius_x0029_>
\
    </soap:Body> \
  </soap:Envelope> \
");
```

Nach dem Eintreffen der SOAP-Response wird die Funktion wsResult aufgerufen:

```
function wsResult() {
  if (http.readystate == 4) {
    // Alle Wurzelknoten ermitteln
    var arr =
      http.responseXML.getElementsByTagName('

    FindNearestMeasure_x005F_x0028_x_x005F_x002C_y_x005F_x002C_
      searchRadius_x005F_x0029_Result');
      createTable(arr);
  }}
```

Die XML-Daten können nun Client-seitig mit JavaScript geparst und verarbeitet werden. Dazu wird in *wsResult* die Funktion *createTable* aufgerufen, welche die Werte unter Verwendung des Document Object Model in eine Tabelle transformiert (Abbildung 30).

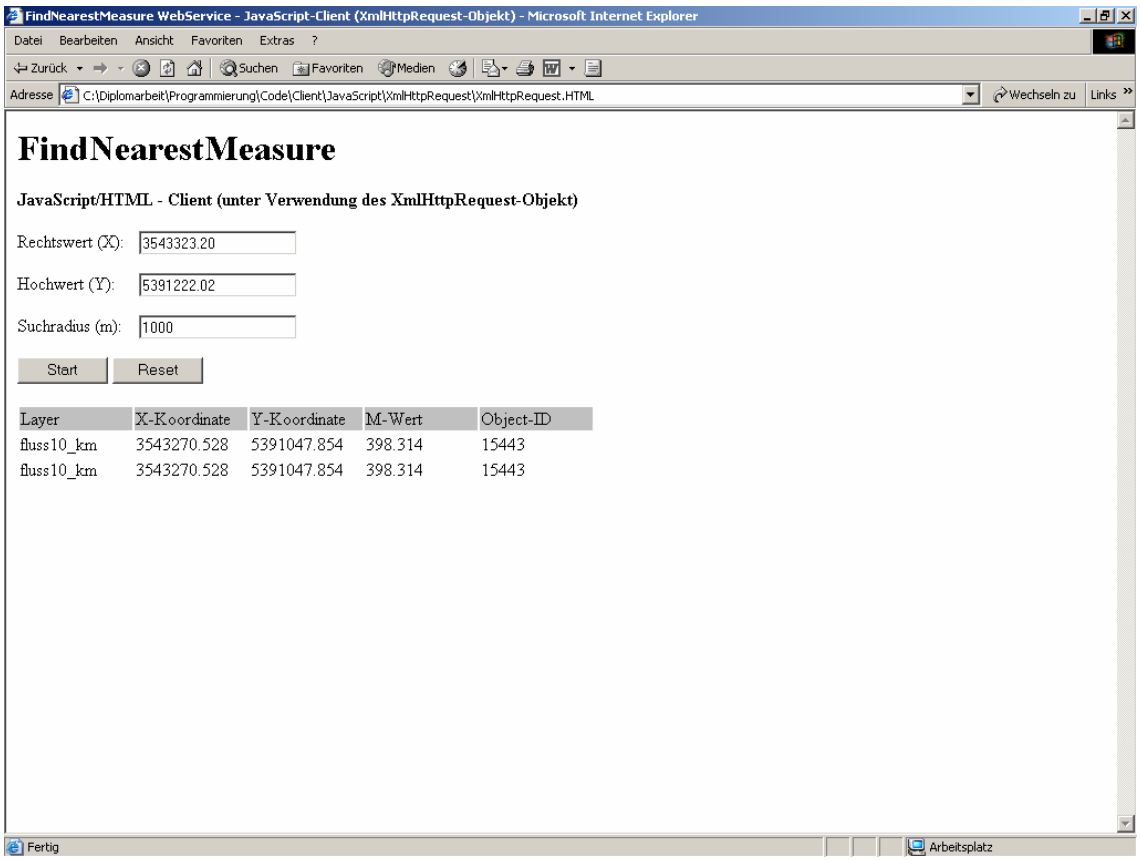

Abbildung 30: HTML-/JavaScript Client

5 Das RIPS-Framework

5.1 Überblick über objektorientierte Wiederverwendbarkeitstechniken

In der objektorientierten Softwareentwicklung existieren verschiedene Ansätze zur Wiederverwendung von bestehendem Programmcode. Ein sehr einfacher Ansatz besteht in der Verwendung so genannter „Code Snippets“. Dabei werden bereits bestehende Code-Fragmente in neue Projekte übernommen. Diese Art der Softwareentwicklung ist jedoch sehr aufwendig und führt bei größeren Projekten schnell zu einem unübersichtlichen Durcheinander. In diesem Kapitel soll ein kurzer Überblick über die wichtigsten Wiederverwendbarkeitstechniken gegeben werden.

5.1.1 Programmbibliotheken

Bibliotheken sind Sammlungen von vorgefertigten Programmteilen, die für die Wiederverwendung in anderen Applikationen vorgesehen sind. Es wird zwischen Quelltextbibliotheken, die Wertedefinitionen, Deklarationen, Klassen usw. beinhalten, sowie den statischen und dynamischen Bibliotheken unterschieden (WEBLEXIKON_2006). Statische Bibliotheken werden direkt nach dem Kompilieren durch einen so genannten Linker mit dem ausführbaren Programm verbunden, wohingegen dynamische Bibliotheken erst bei Bedarf in den Speicher geladen werden. Klassenbibliotheken sind eine besondere Form der Programmbibliotheken und kapseln bestimmte Eigenschaften, Informationen und Mechanismen in einzelne Klassen. In Abbildung 31 ist die typische Gliederung einer Klassenbibliothek nach Pomberger&Blaschek (1996) zu sehen.

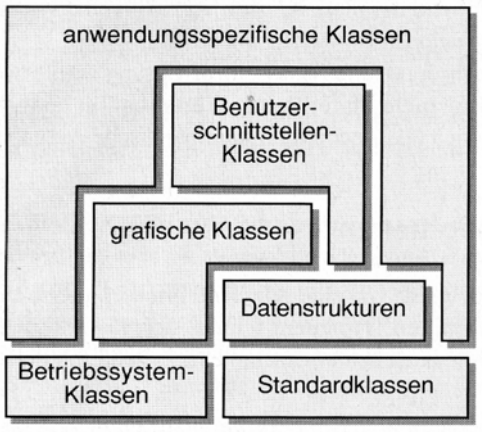

Abbildung 31: Gliederung einer Klassenbibliothek

5.1.2 Komponentenbasierte Softwareentwicklung

Das Ziel dieser Technik ist die Entwicklung von wieder verwendbaren Softwarebausteinen, die jeweils bestimmte Aufgaben erfüllen und nach dem „Plug&Play"-Prinzip zu neuen Anwendungen zusammengesetzt werden. Was genau unter einer solchen Softwarekomponente zu verstehen ist, kann nicht eindeutig geklärt werden, da keine allgemein gültige Definition existiert. Nach Szyperski (1997, S.30) haben Softwarekomponenten die folgenden drei Eigenschaften:

- *"A component is a unit of independent deployment"*
- *"A component is a unit of third-party composition"*
- *"A component has no persistent state"*

Szyperski sieht, wie die meisten anderen Experten auch, die funktionale Unabhängigkeit der einzelnen Komponenten und ihre Fähigkeit, über definierte Schnittstellen mit anderen Bausteinen zu kommunizieren, als die wesentlichen Merkmale. Häufig wird in diesem Zusammenhang die „Legostein-Metapher" angeführt, bei der die einzelnen Komponenten mit der Bauweise von Legosteinen verglichen werden. Jeder Baustein kapselt bestimmte Funktionalitäten, die für das Gesamtsystem zunächst irrelevant sind. Wichtiger sind die Definitionen der Schnittstellen, über welche die einzelnen Elemente zu neuen Systemen zusammengesetzt werden können. CORBA (Common Object Request Broker Architecture), JavaBeans, COM und ActiveX sind bekannte Beispiele für Komponentenmodelle. Ebenso handelt es sich beim .NET-Framework um eine Komponenteninfrastruktur, wobei auch hier der Begriff der Komponente nicht völlig konsistent verwendet wird. Eine sinnvolle Definition liefert Schwichtenberg (SCHWICHTENBERG_2006). Danach ist jede .NET-Assembly, die öffentliche Klassen bereitstellt, als Komponente zu betrachten.

5.1.3 Design Patterns (Entwurfsmuster)

Design Patterns stammen ursprünglich aus der Architektur und kommen in der Entwurfsphase von Softwareprojekten zum Einsatz. Im Wesentlichen handelt es sich bei dieser Technik um die Katalogisierung einmal gefundener Lösungen für die spätere

Wiederverwendung (Kersken, 2005). Entwurfsmuster sind zunächst unabhängig von der Programmiersprache und beschreiben eine erfolgreiche Lösung für wiederkehrende Problemfälle. Der Entwickler kann sich aus einem Musterkatalog den bestmöglichen Ansatz für sein Problem heraussuchen und muss sich dadurch keine Gedanken über die korrekte Entwurfsentscheidung machen. Stattdessen kann er sich direkt auf die Implementierung der Funktionalität konzentrieren. Grundsätzlich besteht ein Muster aus vier Grundelementen:

- *Mustername*
- *Problemabschnitt*
- *Lösungsabschnitt*
- *Konsequenzabschnitt*

Meist sind Muster jedoch noch wesentlich ausführlicher dokumentiert und beinhalten zusätzlich Codebeispiele. Es gibt verschiedene Arten von Mustern. Dazu gehören Entwurfsmuster, Architekturmuster und Idiome.

5.1.4 Frameworks

Nach Pomberger&Blaschek (1996) sind Frameworks “eine besonders auf Wiederverwendbarkeit getrimmte Sammlung von Klassen für ein bestimmtes Anwendungsgebiet“ (S. 260). Im Gegensatz zur reinen Klassenbibliothek oder den Softwarekomponenten stellen Frameworks nicht nur die elementaren Funktionsbausteine zur Verfügung, sondern enthalten zusätzlich die wesentlichen Algorithmen zur Problemlösung. Unter Verwendung von Design Patterns nehmen sie dem Entwickler die wichtigsten Entwurfsentscheidungen ab.
Im Allgemeinen handelt es sich bei einem Framework um ein wieder verwendbares System, das aus fertigen und halbfertigen Subsystemen besteht und dadurch ein Rahmenwerk für eine Vielzahl von Anwendungsfällen bereitstellt. In der Literatur wird ein Framework häufig mit der Konfiguration eines Automobils verglichen. Ein Fahrzeug besteht aus verschiedenen Einzelkomponenten, die für die Funktionsweise unabdingbar sind. Dazu gehören beispielsweise die Karosserie, die Räder oder der

Motor. Deren Funktion und Schnittstellen zu den anderen Komponenten sind vordefiniert. Andere Parameter wie das Aussehen oder das Leistungsvermögen dieser Bauteile können dagegen frei gewählt werden. Auf die objektorientierte Softwareentwicklung übertragen bedeutet dies, dass fertige Subsysteme konkrete Implementierungen darstellen, die der Entwickler instanziieren und zu neuen Applikationen zusammensetzen kann. Halbfertige Subsysteme dagegen repräsentieren Schnittstellen und abstrakte Klassen, die in der eigentlichen Anwendung abgeleitet und umgesetzt werden. Pree (1997) spricht in diesem Zusammenhang von so genannten White-Box und Black-Box Frameworks.
Frameworks kombinieren die Eigenschaften von Klassenbibliotheken, komponentenbasierten Systemen und Design Patterns und bieten derzeit das Höchstmaß an Wiederverwendbarkeit. Durch die Bereitstellung der Anwendungsarchitektur wird der Entwickler von vielen Aufgaben befreit, so dass sich dieser auf die spezifische Anwendung konzentrieren kann und Fehlentscheidungen im Entwurf ausgeschlossen sind. Allerdings ist hierfür eine gute Kenntnis des Entwurfprinzips notwendig.

5.2 Genereller Nutzen einer GIS-Bibliothek

Das wichtigste Kriterium für den Aufbau einer Bibliothek ist die Wiederverwendung bestehender Softwaremodule in neuen Projekten. Daneben ergeben sich zahlreiche weitere Vorteile:

- *Geringerer Entwicklungsaufwand*

  Durch die Einbindung bereits vorhandener Elemente können die Kosten und die Entwicklungszeit für neue Applikationen reduziert werden.

- *Vermeidung von Code-Redundanzen*

  Die zentrale Haltung von Code vermeidet mehrfache Implementierung und vermindert damit den Wartungsaufwand. Unreife Klassen können leichter überwacht werden.

- *Erhöhte Softwarequalität*

  Funktionen müssen nicht für jede Applikation neu entwickelt werden. Stattdessen können die Methoden der Bibliothek verwendet werden, deren Funktionsweise getestet ist und sich bewährt hat.

- *Verringerung des Quellcodeumfangs*

  Der Einsatz von Bibliotheken verringert den Quellcode neuer Programme erheblich und führt zu mehr Übersichtlichkeit.

- *Identische Ergebnisse*

  Verwenden alle Nutzer dieselbe Bibliothek, so erhalten auch alle die gleichen Ergebnisse bei der Durchführung von Geo-Operationen.

- *Plattformunabhängigkeit*

  Die Bibliothek ist auf Basis von .NET plattformunabhängig und kann dadurch in unterschiedliche Clients eingebunden werden. Auch die Wahl der Sprache ist zweitrangig.

- *Zentrale Änderungen und Ergänzungen*

  Durch die zentrale Haltung wirken sich Änderungen und Ergänzungen einer Klasse direkt auf alle Klassen aus, welche diese verwenden oder von ihr abgeleitet sind. Neue Funktionen können ohne größeren Aufwand hinzugefügt werden.

- *Einheitlicher Code*

  Unter Einhaltung von Coding Conventions wird der Programmcode einheitlich und damit transparent für alle Entwickler.

- *Vereinfachtes Deployment*

  Klassenbibliotheken lassen sich leicht auf anderen Arbeitsplätzen installieren. Ebenso ist der Zugriff über Internet oder Netzwerk möglich.

Neben den Vorteilen für den Entwickler ist die Wiederverwendbarkeit aber auch eine Pflicht gegenüber dem Anwender, der seine Software nicht mit jeder Änderung anpassen möchte. Es ist daher wichtig, das Framework gleich zu Beginn so zu gestalten, dass nachträgliche Erweiterungen oder Änderungen vom Endanwender ohne größeren Aufwand übernommen werden können. Der Einsatz von Klassenbibliotheken bringt also viele Erleichterungen mit sich. Da sie aber selbst zu ausgewachsenen Softwareprodukten werden können, die verwaltet und gewartet werden müssen, ergeben sich auch einige Nachteile:

- *Übersichtlichkeit*

  Mit zunehmender Komplexität verringert sich die Übersichtlichkeit der Bibliothek

- *Unterschiedliche Programmierstile*

  Die unterschiedlichen Programmierstile der Entwickler müssen durch einheitliche Coding Conventions aufeinander abgestimmt werden.

- *Hoher Einarbeitungsaufwand*

  Je größer die Bibliothek ist, desto mehr Kenntnis wird vom Entwickler abverlangt. Das ist oftmals mit größerem Aufwand verbunden, als die benötigten Funktionen neu zu programmieren.

- *Hoher Dokumentationsaufwand*

  Eine gut organisierte Bibliothek muss gründlich dokumentiert werden, um die Übersichtlichkeit zu gewährleisten und auch neuen Entwicklern einen schnellen Zugang zu ermöglichen. Daher sollte auch die Dokumentation einheitlichen Standards unterliegen.

- *Ausgereiftes Konzept notwendig*

  Ohne ausreichend durchdachte Grundkonzeption ist eine Klassenbibliothek relativ wertlos. Es muss klar geregelt sein, wie neue Funktionen eingegliedert werden können.

5.3 Verwaltung der Bibliothek

Da die Bibliothek in den meisten Fällen von vielen verschiedenen Programmierern gleichzeitig entwickelt wird, muss unbedingt dafür gesorgt werden, dass überall die aktuellste Version verwendet wird. Das ist insbesondere bei vernetzten Arbeitsplatzrechnern nicht ganz einfach zu bewerkstelligen. Pomberger&Blaschek (1996) schlagen deshalb den Einsatz eines „Klassenbiliothekars“ (Code-Master) vor, der als zentrale Schaltstelle agiert und die Verteilung der jeweils aktuellsten Version sowie die Pflege der Bibliothek überwacht. Zu seinen Aufgaben zählen:

- *Die Aufnahme neuer Klassen*

  Neue Klassen und Funktionen können nicht einfach von jedem Entwickler in die Bibliothek eingepflegt werden. Der Code-Master muss überprüfen, ob die Klasse in das Konstrukt der Bibliothek passt und ob sie ausreichend dokumentiert ist. Die Aufnahme neuer Klassen muss ebenfalls dokumentiert und allen beteiligten Programmierern mitgeteilt werden.

- *Schaffung von Richtlinien*

  Dazu gehören z.B. die Coding Conventions, die Dokumentationsrichtlinien, die Strukturierung von Methoden und die Konventionen über Sichtbarkeitsregeln und Zugriffsrechte. Als Grundlage für diese Richtlinie kann z.B. die Kernbibliothek gelten (.NET-Base Class Library).

- *Überwachung unreifer Klassen und Änderungskontrolle*

  Neue Klassen müssen überwacht und ihre Funktionsweise getestet werden. Dazu ist die Rückmeldung von den Nutzern notwendig. Auch Korrekturen fehlerhafter Klassen müssen genau dokumentiert werden.

- *Kooperation beim Klassenentwurf*

  Der Entwurf neuer Klassen sollte in Abstimmung mit dem Code-Master erfolgen, damit die gemeinsamen Richtlinien eingehalten werden.

5.3.1 Coding Conventions

Coding Conventions sind ein Regelsatz, nach welchen der Quelltext einer Programmiersprache formatiert wird. Ziel von Coding Conventions ist es, die Konsistenz und Lesbarkeit von Quellcode zu verbessern und damit die Integration neuer Software sowie deren Wartung zu vereinfachen. Programmierer sollten sich an diese Konventionen halten, um anderen Entwicklern den Einstieg in den Quellcode so einfach wie möglich zu gestalten. Bei der Entwicklung von Klassenbibliotheken kann es jedoch manchmal Situationen geben, in denen es durchaus sinnvoll ist, das Regelwerk außer Kraft zu setzen. Dies sollte aber nur in Absprache mit dem Code Master geschehen.
Coding Conventions umfassen Elemente der optischen Lesbarkeit wie die Positionierung von Syntaxelementen (z.B. `{}`, `[]`, `()`, …), Einrückungsstil, der Einsatz von Kommentaren und deren Formatierung, Namenskonventionen, Verschachtelungstiefe sowie den Umgang mit Leerezeichen und Leerzeilen. Daneben müssen auch semantische Aspekte wie die Vermeidung von Code-Redundanzen, die Vorgehensweise beim Exception-Handling, die Verwendung von Entwurfsmustern, die Wiederverwendbarkeit und die Robustheit der Anwendung berücksichtigt werden. Als Vorbild können die von Microsoft definierten Design Guidelines für das .NET-Framework dienen (MSDN_2006). Sun stellt auf seiner Internetseite Richtlinien für die Quellcode-Codierung mit Java auf (JAVACC_2006). Weitere Empfehlungen für die Quellcodeformatierung für C# finden sich außerdem auf den Seiten von Lance Hunt (HUNT_2006).
Mit Hilfe von so genannten „Style Checker“- und „Beautifier“-Tools kann die Einhaltung von Coding Conventions direkt aus der Entwicklungsumgebung heraus überprüft und gegebenenfalls korrigiert werden. Diese Quelltextformatierungsprogramme (STYLECHECK_2006) eignen sich hauptsächlich für die Überprüfung der optischen Lesbarkeit. Sie bieten aber teilweise auch schon die Möglichkeit, die Einhaltung der Semantik zu untersuchen.

5.3.2 Verwendung externer Tools

Für viele Anwendungsbereiche stehen dem Entwickler bereits fertige Tools zur Verfügung. Um beispielsweise die Funktionsweise neuer oder noch fehlerhafter

Klassen zu testen, können Bibliotheken wie das Logging-Framework „log4net“, (APACHE_2006) eingesetzt werden. Logging-Frameworks sind vor allem dann von Bedeutung, wenn Software bereits ausgeliefert ist und nicht mehr nur mit dem Debugger überprüft werden kann. Fehlermeldungen werden unter Vergabe von Prioritäten an das Logging-System übergeben, welches die log-Messages an das gewünschte Zielmedium (log-File, Standardausgabe, …) sendet.
Eine weitere Bibliothek ist das .NET Zip Library, das Klassen und Methoden zur Komprimierung und Dekomprimierung von Zip-Archiven bereitstellt. Die Bibliothek ist als dll-Datei kompiliert und vollständig in C# programmiert (#ZIPLIB_2006).

5.4 Systemarchitektur des RIPS-Framework

Wie in Abbildung 32 zu sehen ist, wird das RIPS-Framework in zwei Teilsysteme „Web Service“ und „Desktop“ untergliedert. Die Gründe hierfür liegen in der unterschiedlichen Programmierung mit der ArcObjects-Bibliothek und dem Deployment der beiden Frameworks. Im Gegensatz zur Desktop-Programmierung, bei der direkt mit generischen ArcObjects gearbeitet wird, muss sich der Entwickler bei der Erstellung von ArcGIS Server-Anwendungen an die Vorgaben der .NET ADF-Laufzeitumgebung halten. Die Programmierung mit der Server API wird auch als „remotely programming ArcObjects“ bezeichnet (ESRI, 2004, S.76). D.h. es wird nicht unmittelbar mit den auf dem Server befindlichen ArcObjects programmiert, zumal die .NET ADF gar keine ArcObjects-Komponenten beinhaltet. Stattdessen stellt sie über Assemblies Proxies bereit, über die auf ArcObjects des Servers zugegriffen werden kann.
Zwar sind die Unterschiede in der Programmierung (vgl. Kapitel 4.3.3) nicht sehr groß, dennoch erscheint es schon aufgrund der unterschiedlichen Art der Auslieferung als sinnvoll, zwei voneinander getrennte Frameworks zu entwickeln. Das Desktop Framework muss lokal für 1-n Clients installierbar sein und eventuell eine Installationsroutine mit Dokumentation umfassen. Das Web Service Framework dagegen bietet zentral abgelegte Geofunktionen, die über das Netzwerk oder über das Internet verfügbar sind. Einer der klaren Vorteile darin liegt in der Vermeidung von aufwendigen Softwareinstallationen auf Seiten des Klienten. Browser oder Desktop-Anwendungen können über Web Services auf die Funktionen des GIS Servers zugreifen, ohne dass Veränderungen an der Software vorgenommen werden müssen.

Dennoch steht dem Anwender die umfassende GIS-Funktionaliät eines Desktop-GIS zur Verfügung. Neben dem ArcGIS Server soll das Web Service Framework auch den Zugriff über ArcIMS und Oracle Spatial ermöglichen, da diese Systeme in einigen Fällen leistungsfähiger sind.

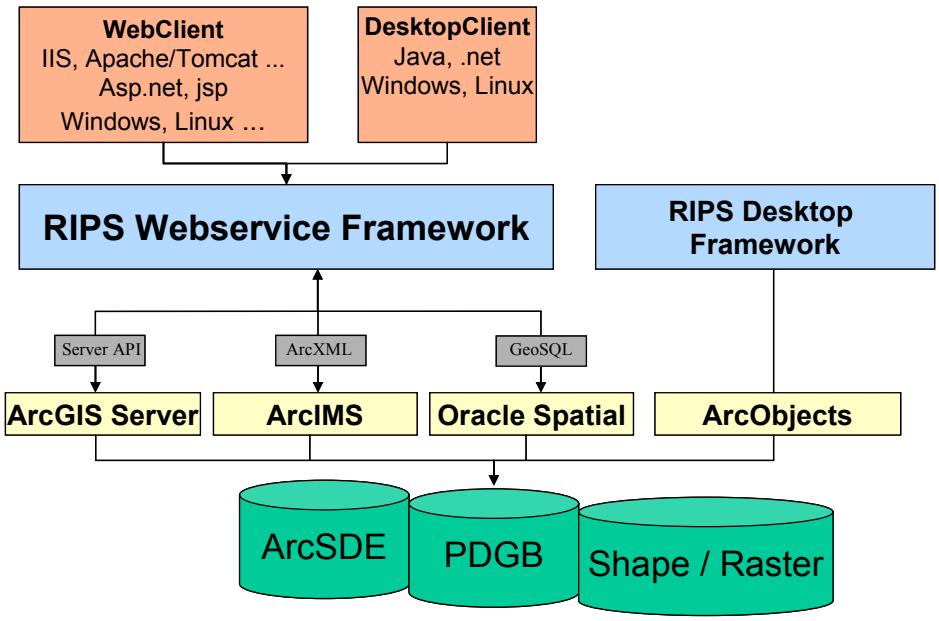

Abbildung 32: Systemarchitektur des RIPS-Framework

5.5 RipsDesktop

Im Rahmen von RipsDesktop soll die von der ArcView-Erweiterung ArcWaWiBo bereitgestellte Funktionalität auf Basis von C# .NET nach ArcGIS migriert werden. Ziel ist es, der Rips-Entwicklerrunde und den WAGBIS-Anwendern Erweiterungen bereitzustellen, die je nach Bedarf den ArcGIS-Applikationen hinzugeladen werden können. Da den einzelnen Nutzergruppen unterschiedliche Datentöpfe zur Verfügung stehen, sollte das System hinsichtlich der Interoperabilität zu den jeweiligen Quellen skalierbar sein. Die Funktionalität muss unabhängig davon, ob eine Shape-Datei, eine Personal Geodatabase (PGDB) oder eine ArcSDE-Datenbank verwendet wird, ausgeführt werden können. Wie erwähnt, bestehen an das Desktop-Framework andere Anforderungen bezüglich des Deployment und der Wartung als an das Web Service Framework. Während die Web Service-Variante durch die zentrale Haltung permanent

gepflegt und erweitert werden kann, ist die Auslieferung von Verbesserungen am Desktop-Framework nur in Intervallen möglich.

5.6 RipsWeb

5.6.1 Basisarchitektur

Zur Zielgruppe des Web Service Frameworks gehören die RIPS- und die WAGBIS-Entwicklerrunde sowie externe Anwender. Abbildung 33 zeigt den Aufbau von RipsWeb. Konfigurationsparameter, die für den Zugriff auf Datenquellen benötigt werden, sind grundsätzlich als *key/value*-Paare in XML-basierten Anwendungskonfigurationsdateien abgelegt. Durch die Trennung von der Applikationslogik können die Datenquellen geändert werden, ohne dass der Quellcode modifiziert werden muss.

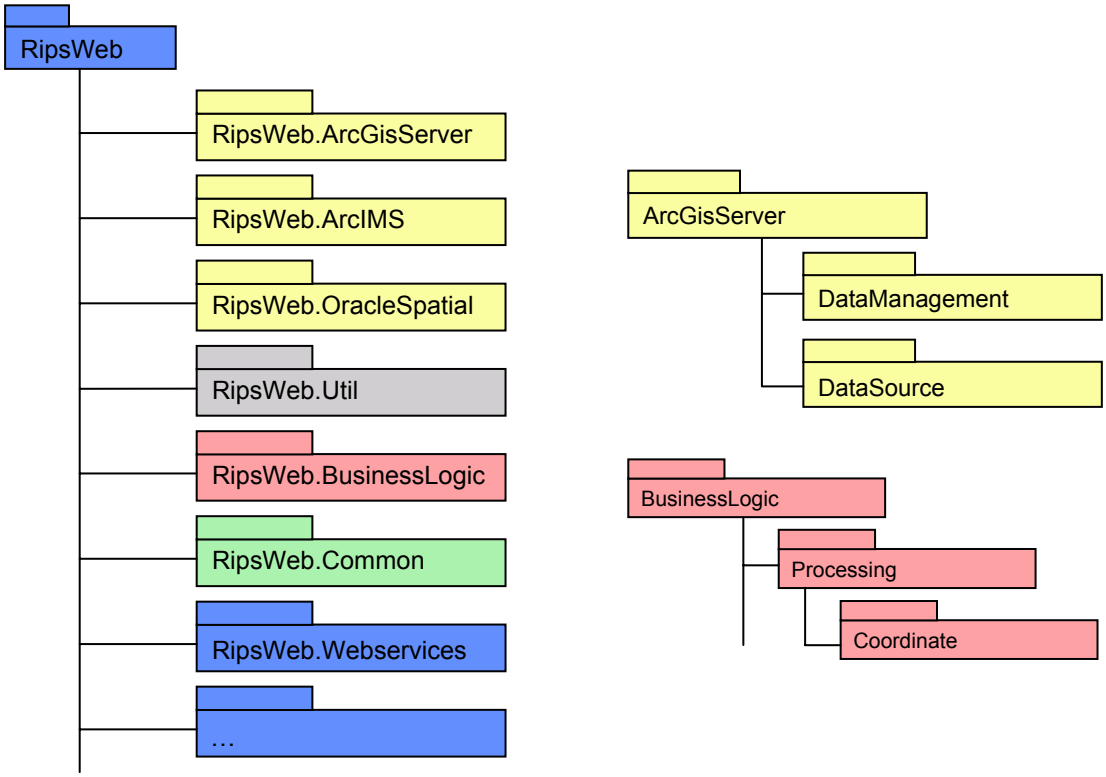

Abbildung 33: Aufbau von RipsWeb

5.6.2 Namespaces

RipsWeb.ArcGisServer

Der *ArcGisServer*-Namespace dient der Datenverarbeitung über ArcGIS Server. *DataManagement* bietet Klassen für die Verarbeitung von Feature- und Rasterdaten und baut über die Klassen in *DataSource* eine Verbindung zur entsprechenden Datenquelle auf (Raster, Shape, Sde, Access). Dazu wird über eine Factory-Klasse eine Workspace-Instanz an die aufrufende Methode zurückgeliefert (Abbildung 34). Die Verbindung zum GIS Server und der Aufbau von ServerContexts erfolgt über den Namespace *System* (Abbildung 35).

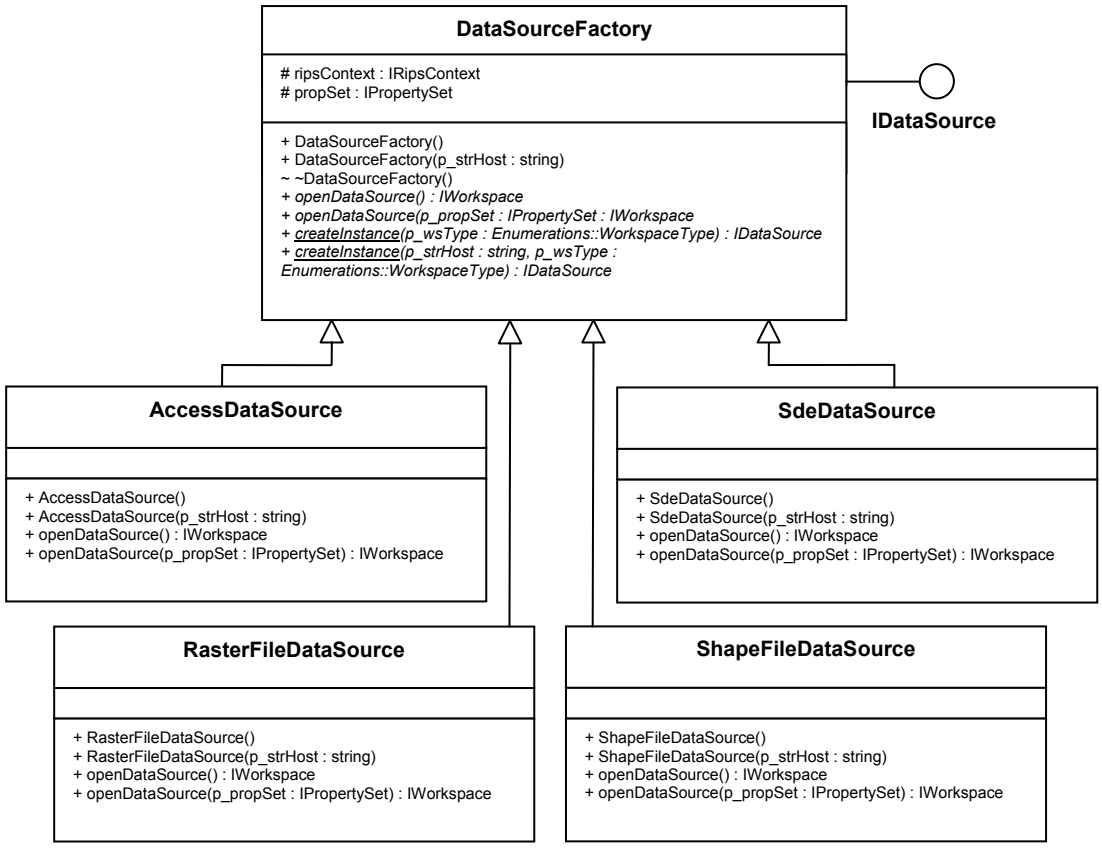

Abbildung 34: Klassendiagramm DataSource

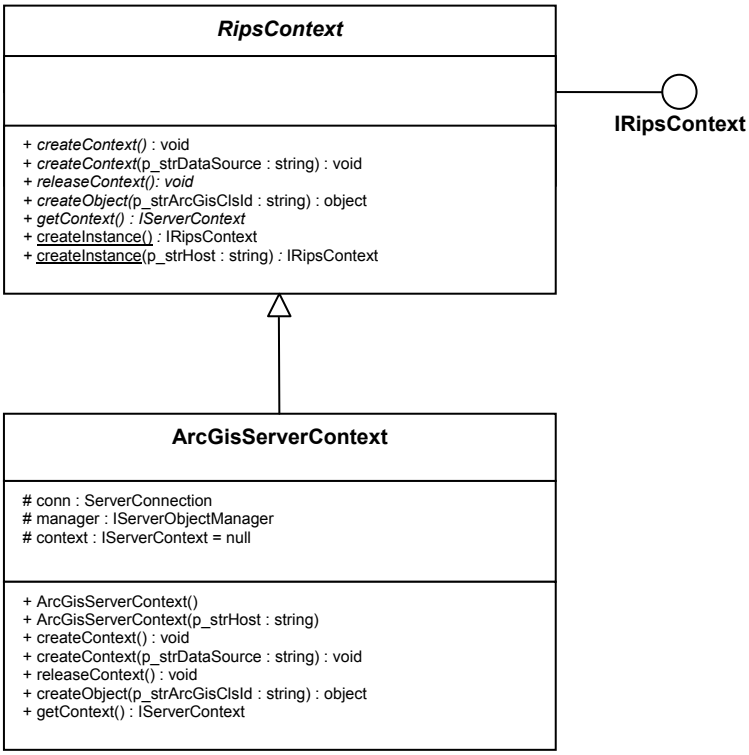

Abbildung 35: Klassendiagramm System

RipsWeb.ArcIms und RipsWeb.OracleSpatial

Die beiden Namensräume sind analog zum ArcGisServer-Namespace und dienen der Datenverarbeitung über ArcIMS und Oracle Spatial.

RipsWeb.BusinessLogic

Dieser Namespace soll den allgemeinen Teil der Framework-Funktionalität enthalten, die nicht über ArcObjects realisiert wird.

RipsWeb.Util

Beinhaltet Helferklassen, Definitionen von Konstanten und Enumerationen. Hier könnten auch Klassen zur Erzeugung von Datenstrukturen abgelegt werden, die von Web Services an den Client zurückgeliefert werden.

RipsWeb.Common

Enthält Interface-Definitionen, die in mehreren Namespaces verwendet werden.

RipsWeb.Webservices

Hier werden die Web Service-Klassen verwaltet. Die Funktionen werden soweit gekapselt, dass die Web-Methoden nur noch die Methoden der RipsWeb-Assembly aufrufen. Dort erfolgen der Verbindungsaufbau zur Datenquelle und die Ausführung der Geo-Funktionen.

5.6.3 Kategorisierung der Web Services

Eine der großen Schwierigkeiten von Service-orientierten Architekturen besteht in der sinnvollen Gruppierung der Dienste, so dass sie problemlos aufgefunden und neue Funktionen eindeutig zugeordnet werden können. Um eine geeignete Schubladenstruktur zu finden, wurden verschiedene Ansätze untersucht. Größere Web Service-Verzeichnisse wie die öffentlichen UDDI (UDDI_2006) oder die ESRI ArcWeb Services (ESRI_2006b) bieten ein sehr breites Spektrum an Funktionalität und klassifizieren nach Branchen oder nach geographischen Gesichtspunkten. UDDI wird häufig mit den „Gelben Seiten“ verglichen. Hieran wird die Problematik deutlich: Ein Bäckereibetrieb könnte sowohl unter „Bäckereien“ als auch unter dem Oberbegriff „Konditoreien“ auffindbar sein. ESRI kategorisiert seine Dienste nach „Business Information“, „Demographics“, „Geographic Boundaries & Landmarks“, „Imagery“, „Physical Environment“, „Transportation“, „Weather“ und unterteilt zudem nach geographischen Regionen.
Neben diesen Web Service-Verzeichnissen wurden auch GIS-Desktop-Applikationen zum Vergleich herangezogen. Das von der Firma WASY entwickelte System ArcWFD gruppiert seine Werkzeuge nach „Edit“, „Auskunft“, „Analyse“ und „Bericht“ und orientiert sich damit an den grundlegenden GIS-Funktionalitäten (Abbildung 36).

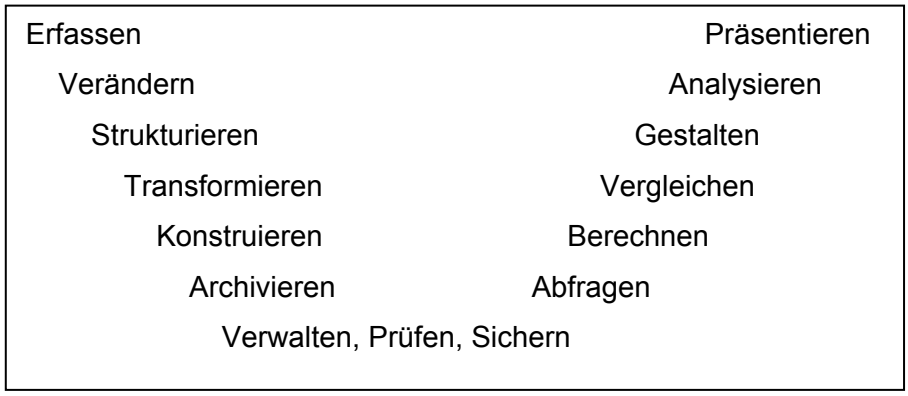

Abbildung 36: Basisfunktionen eines GIS

Eine weitere Möglichkeit wäre die Definition von Schubladen für die speziellen Funktionen der einzelnen Fachmodule. So finden sich bei der ArcView-Erweiterung ArcWaWiBo die Kategorien „Basic“, „Karto“, „ALK“ und „WAABIS“ (Abbildung 4).
Da das Web Service-Framework letztendlich einen Pool an Grundfunktionalitäten für die Verwendung durch die RIPS-Entwicklerrunde und die Anwender der Fachmodule bereitstellen soll, erscheint eine Struktur in Anlehnung an die ESRI ArcToolbox (Abbildung 37) am sinnvollsten. Sie spiegelt die grundlegenden GIS-Operationen wider und bietet dadurch ein solides Fundament für spätere Erweiterungen. Da sich die Entwicklung der Toolbox bereits über mehrere Versionen erstreckt, kann davon ausgegangen werden, dass die Erfahrungswerte vieler Jahre in den Aufbau der Toolbox eingearbeitet wurden. Ein weiterer Vorteil besteht darin, dass ArcObjects-Programmierer mit der Hierarchie vertraut sind und sich nicht erst in eine neue Struktur hineindenken müssen.

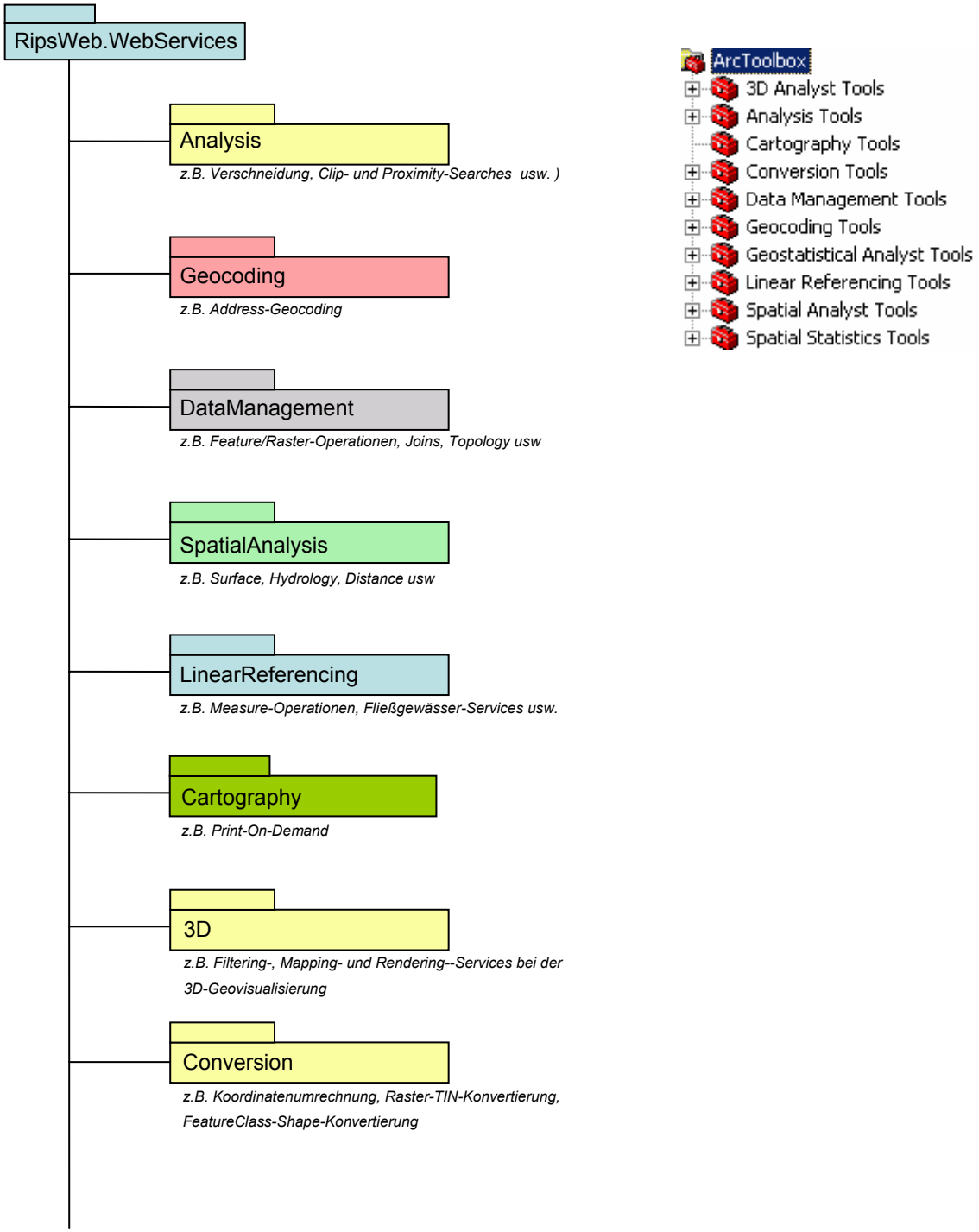

Abbildung 37: ArcToolbox und die Klassifizierung der RipsWeb-Web Services

Prinzipiell wäre auch eine Klassifizierung nach Objektarten denkbar. Da es im RIPS-Umfeld jedoch sehr viele unterschiedliche Objekte gibt und bislang noch nicht abschließend geklärt ist, welche Funktionen in das Framework integriert werden sollen, ist eine derartige Gruppierung ungeeignet.

5.6.4 Benennung der Web Service-Klassen

Die Web Service-Klassen sollten mit einem prägnanten Begriff benannt werden. Wie im Beispiel der Abbildung 38 zu sehen ist, tragen sie idealerweise den Namen des Objektes, das sie zurückliefern. Dadurch können die Methodennamen kurz gehalten werden. Es ist allerdings zu prüfen, inwieweit diese Logik beim Hinzufügen weiterer Funktionen beibehalten werden kann.

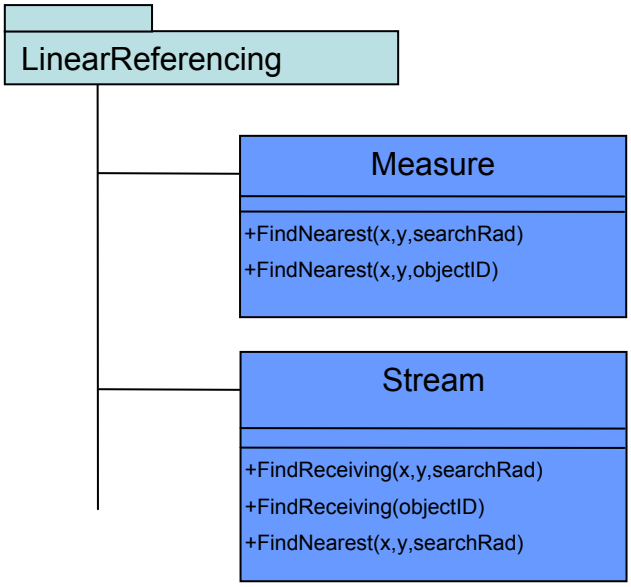

Abbildung 38: Benennung der Web Service-Klassen

6 Veröffentlichung und Erweiterung von RipsWeb

6.1 Veröffentlichung der Web Services

Neben der Klassifizierung stellt sich die Frage, wie die Web Services des RipsWeb Framework den Anwendern für die Einbindung in ihre Softwarelösungen verfügbar gemacht werden können. Der Nutzer muss über die Existenz von Diensten informiert werden, und er muss wissen, wie er sich Zugriff darauf verschaffen kann. Anstelle jeden Service isoliert bereitzustellen, bietet sich die Einrichtung zentraler Zugangspunkte an, die auf alle verfügbaren Dienste verweisen. Zum Veröffentlichen von Web Services stehen verschiedene Technologien zur Verfügung. Dazu gehören das bereits erwähnte UDDI (vgl. Kapitel 3.1.3), die Discovery-Dateien und der WS-Inspection-Standard.

6.1.1 UDDI-Verzeichnisse

Es wird zwischen öffentlichen, privaten und halbprivaten Verzeichnisdiensten unterschieden. Private Verzeichnisse dienen ausschließlich der Anwendung im firmeneigenen Intranet, während halbprivate Verzeichnisse zusätzlich externen Partnern den Zugriff auf Dienste ermöglichen sollen. Die Registrierung in öffentlichen UDDI-Verzeichnissen ist immer dann sinnvoll, wenn die Web Services einem größtmöglichen Anwenderkreis angeboten werden sollen.
UDDI-Services besitzen eine SOAP-Schnittstelle und sind selbst als Web Services realisiert, die Metadaten über andere Webdienste bereitstellen. Sie sind in erster Linie darauf ausgelegt, programmgesteuert eingesetzt zu werden. Es gibt aber auch Browser-basierte Schnittstellen, damit Entwickler manuell nach Services suchen können. Öffentliche UDDI-Verzeichnisse werden derzeit von den Gründungsmitgliedern des UDDI-Konsortiums Microsoft, Ariba, IBM sowie von SAP betrieben (UDDI_2006). Die UDDI-Technologie befindet sich zwar immer noch im Entwicklungsstadium, wird sich aber wahrscheinlich aufgrund der Unterstützung durch viele bedeutende Unternehmen langfristig als Standard durchsetzen.
Für größere Unternehmen besteht die Möglichkeit, eigene UDDI Server zu betreiben. So beinhaltet beispielsweise der Windows Server 2003 mit Enterprise UDDI Services

ein integriertes Feature, um Registrierungsstellen auf Basis der UDDI- Spezifikationen einzurichten. Mit dem UDDI SDK stellt Microsoft zudem ein Software Development Kit zur Verfügung, mit dem Client-Applikationen erstellt werden können, welche die Funktionen des UDDI Servers nutzen. Das Kit besteht aus einer Dokumentation, der Microsoft.UDDI-Assembly für .NET (Abbildung 39) und verschiedenen Beispiel-Applikationen. Nähere Informationen finden sich in der MSDN Library (MSDN_2006).

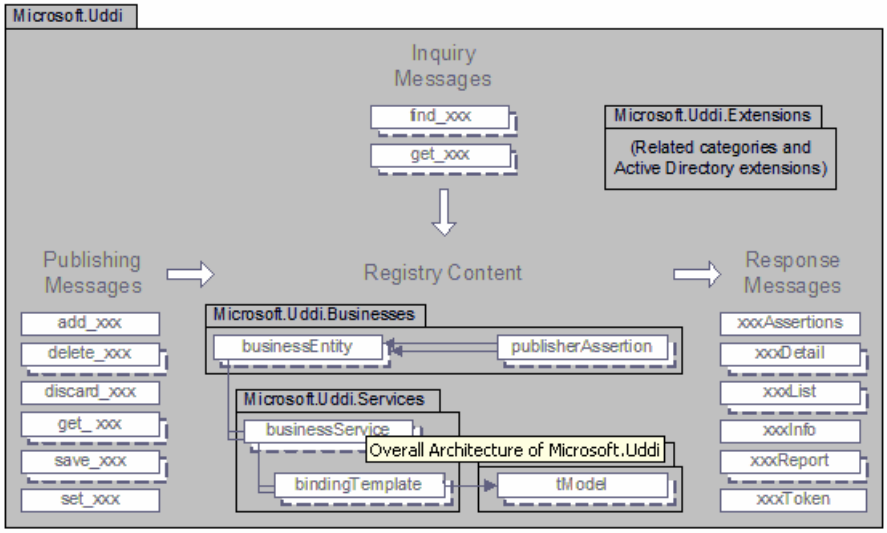

Abbildung 39: Architektur der Microsoft.Uddi-Assembly für .NET

6.1.2 Discovery-Dateien

Nicht in jedem Fall wird gleich ein UDDI-Verzeichnis zum Verbreiten von Web Services benötigt. Da der Betrieb dieser Dienste sehr aufwendig ist, lohnt sich UDDI nur für größere Firmen. DISCO-Dateien sind eine proprietäre Lösung von Microsoft und in vielen Fällen ausreichend, um XML-Dienste kleineren Nutzerkreisen zur Verfügung zu stellen. Gegenüber UDDI sind sie wesentlich einfacher in der Handhabung und für die Zwecke des RipsWeb vermutlich ausreichend. Discovery-Dateien ermöglichen dem Client, auf die Services eines Web Servers zuzugreifen. Es handelt sich dabei um XML-Dokumente, die URL-Verweise auf andere DISCO-Dateien oder WSDL-Beschreibungen enthalten können. Um alle Dienste des RipsWeb unter einer einzigen URL anzubieten, kann auf dem Web Server z.B. eine Datei „RipsServices.disco“

eingerichtet werden. Dazu wird dem Web Service-Projekt in Visual Studio eine statische Discovery-Datei hinzugefügt (Abbildung 40).

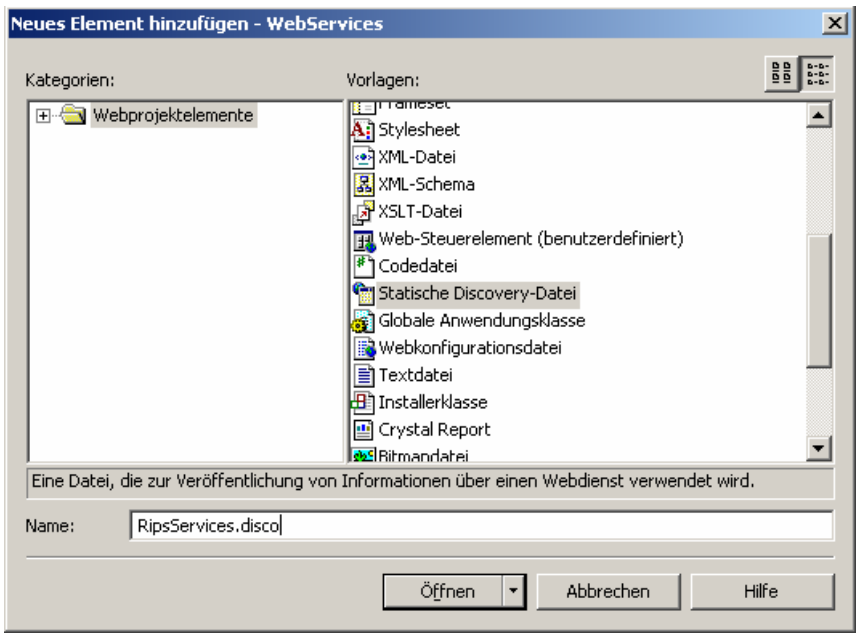

Abbildung 40: Hinzufügen einer statischen Discovery-Datei in Visual Studio

Der XML-Code der Datei sieht folgendermaßen aus:

```
<?xml version="1.0" encoding="utf-8"?>
<discovery xmlns="http://schemas.xmlsoap.org/disco/">
  <contractRef
   ref="http://localhost/RipsWeb/src/RipsWeb/Webservices/
   LinearReferencing/Measure.asmx?wsdl"
   docRef="http://localhost/RipsWeb/src/RipsWeb/Webservices/
   LinearReferencing/Measure.asmx "
   xmlns="http://schemas.xmlsoap.org/disco/scl/"/>

  <contractRef
   ref="http://localhost/RipsWeb/src/RipsWeb/Webservices/
   LinearReferencing/Stream.asmx?wsdl"
   docRef="http://localhost/RipsWeb/src/RipsWeb/Webservices/
   LinearReferencing/Stream.asmx"
   xmlns="http://schemas.xmlsoap.org/disco/scl/"/>
</discovery>
```

Die Datei verwendet ein root-Element *<discovery>* aus dem Namespace *http://schemas.xmlsoap.org/disco/.* Mit dem *<contractRef>*-Element wird eine Referenz zum WSDL-Dokument anderer Web Services erstellt. Zusätzlich ist es möglich, über das *docRef*-Attribut auf die Dokumentation des Dienstes zu verweisen. Nun sind alle referenzierten Services unter der URL http://[Webserver]/.../RipsServices.disco erreichbar und können von der Discovery-Seite aus in Softwareprojekte eingebunden werden (Abbildung 41).

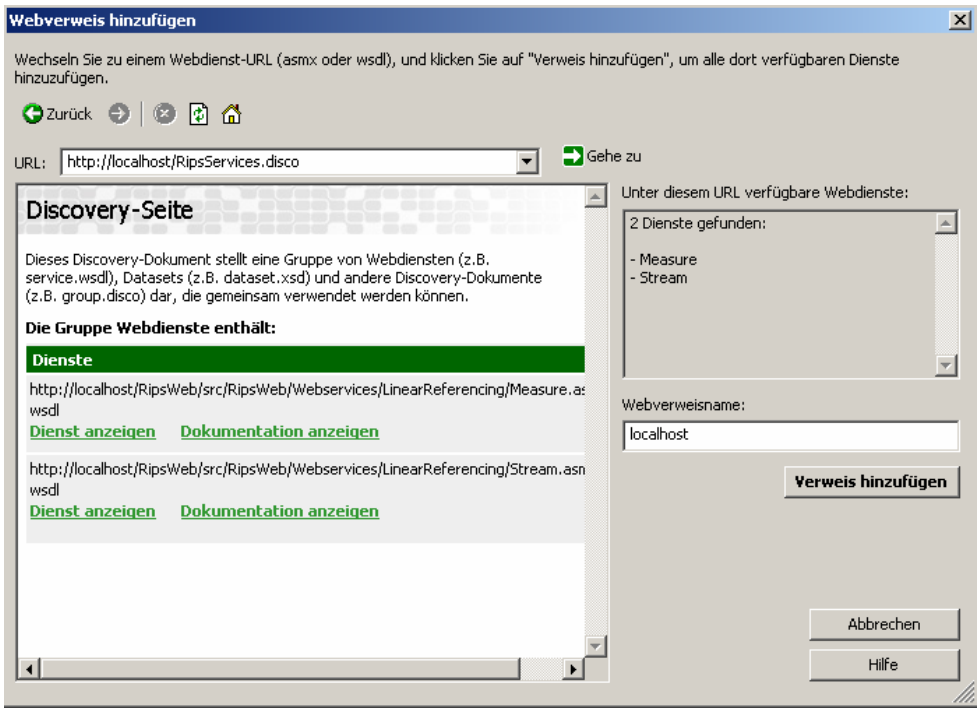

Abbildung 41: Web Reference auf eine disco-Datei

ASP.NET bietet auch die Möglichkeit, dynamische Discovery-Dateien zu erstellen. Bei Aufruf eines solchen Files (*.vsdisco) durchsucht ASP.NET den Web Server (bzw. das Verzeichnis, in dem die disco-Datei abgelegt ist) und generiert dynamisch eine DISCO-Datei mit Referenzen auf alle verfügbaren Dienste. Dieses Feature ist jedoch standardmäßig abgeschaltet, da es den Server sehr stark belasten kann. Microsoft empfiehlt, dynamische Discovery-Dateien nicht auf einem Produktionsserver zu aktivieren. Deren Einsatz sollte daher genau überlegt werden (Freeman&Jones, 2003 und Fraser&Livingstone, 2002).

6.1.3 Web Service Inspection

WS-Inspection (MSDN_2006) ist ein weiterer Standard von Microsoft und IBM für das Auffinden von Web Services und verfolgt einen zu UDDI völlig unterschiedlichen Ansatz. Während UDDI auf einigen wenigen zentralen Verzeichnissen basiert (Abbildung 42), setzt WS-Inspection auf viele kleine und dezentralisierte Verzeichnisse, in denen nur wenige Anbieter ihre Dienste veröffentlichen (Dostal et al, 2005).

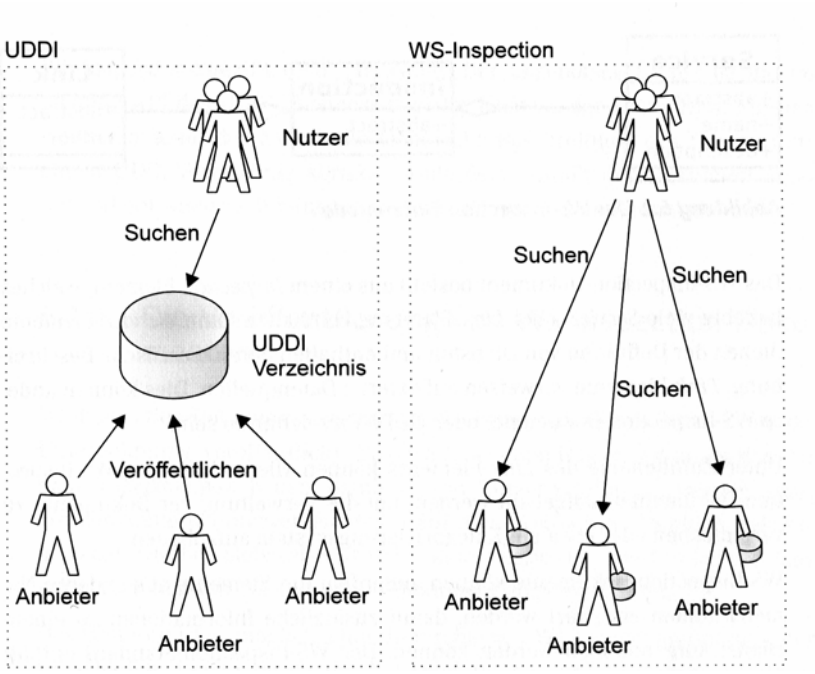

Abbildung 42: Vergleich von UDDI und WS-Inspection

Bei WS-Inspection handelt es sich, wie bei den Discovery-Dateien, um eine vergleichsweise einfache Lösung. Der Standard eignet sich besonders dann, wenn der Dienste-Anbieter dem suchenden Konsumenten bekannt ist und der Service lediglich genau lokalisiert werden muss. WS-Inspection ist aber nicht als direkter Konkurrent zu UDDI anzusehen, sondern kann komplementär als Bindeglied zwischen den bestehenden Standards und dem Web Service Client eingesetzt werden (XML&WS_2003). WS-Inspection basiert auf XML-Dokumenten, die dem Anwender unter Nutzung des HTTP-Protokolls eine Liste mit den verfügbaren Web Services und deren WSDL-Beschreibungen bereitstellen. Der Aufbau dieser XML-Dokumente ist in der WS-Inspection-Spezifikation definiert (Abbildung 43).

Abbildung 43: WS-Inspection-Datenmodell

Das WS-Inspection-Dokument besteht aus einem root-Tag <*inspection*>, welches beliebig viele <*service*>- und <*link*>- Elemente enthalten kann. Die <*service*>-Tags definieren die einzelnen Dienste und enthalten deren technische Beschreibung, während die <*link*>-Tags auf externe Datenquellen verweisen. Das können wiederum andere WSIL-Dokumente, UDDI-Verzeichnisse oder WSDL-files sein. Mit Hilfe des <*link*>-Elementes können Hierarchien und Kategorisierungssyteme erzeugt werden. Das folgende Listing zeigt den Aufbau eines WSIL-Dokuments:

```
<?xml version="1.0" encoding="utf-8" ?>
<inspection xmlns=…
  …hier können mehrer <abstract/>,<services/>und <link/> stehen
  <service>
    <abstract>
      …einfache Service-Dokumentation
    </abstract>
    <description>
      …z.B. ein ServiceKey von UDDI
    </description>
  </service>
  <link referencedNamespace=…>
    …Verweise auf andere Service-Beschreibungen
  </link>
</inspection>
```

Um Web Service Clients das Auffinden von WSIL-Dokumenten zu ermöglichen, sieht die Spezifikation zwei Möglichkeiten vor. Die erste ist die Einrichtung einer Datei „inspection.wsil“ am zentralen Einstiegspunkt einer Web-Präsenz bzw. im Root-Verzeichnis des Web Servers. Der Aufruf dieser Datei liefert die XML-basierte Auflistung aller verfügbaren Dienste des Verzeichnisses. Als Beispiel hierfür sei der im folgenden Kapitel vorgestellte Verzeichnisdienst „xmethods“ genannt. Die zweite Möglichkeit besteht in der Einbettung der URLs von WSIL-Dokumenten in die *meta*-Tags des HTML-Dokuments.

6.1.4 Browser-basierte Verzeichnisse

Um Web Service-Verzeichnisse nicht nur programmatisch aufrufbar, sondern auch für den menschlichen Nutzer zugänglich zu machen, sollte die Einrichtung einer HTML-basierten Lösung in Erwägung gezogen werden. Auch die meisten UDDI-Verzeichnisse stellen solche Internetseiten bereit. Im Folgenden wird ein Ansatz untersucht, der als Vorlage für RipsWeb dienen könnte.

Xmethods (XMETHODS_2006)

Diese Seite (Abbildung 44) enthält Metadaten zu zahlreichen Diensten aus den unterschiedlichsten Bereichen. Außerdem existieren verschiedene Schnittstellen (UDDI, SOAP, RSS, WS-Inspection, DISCO) für den programmatischen Zugriff. Die tabellarische Auflistung der Services enthält Angaben zum Anbieter und Style, den Servicenamen, eine Beschreibung sowie die Plattform, auf deren Basis der Service implementiert wurde. Darüber hinaus ist ein Online Tool zum Testen der Funktionen integriert. Wird auf den Namen des Dienstes geklickt, öffnet sich ein Dokument mit einer detaillierten Erläuterung (Abbildung 45). Auf dieser Seite finden sich zudem Binding-Informationen und der URL des Endpunktes, über den der Service in das eigene Projekt integriert werden kann.

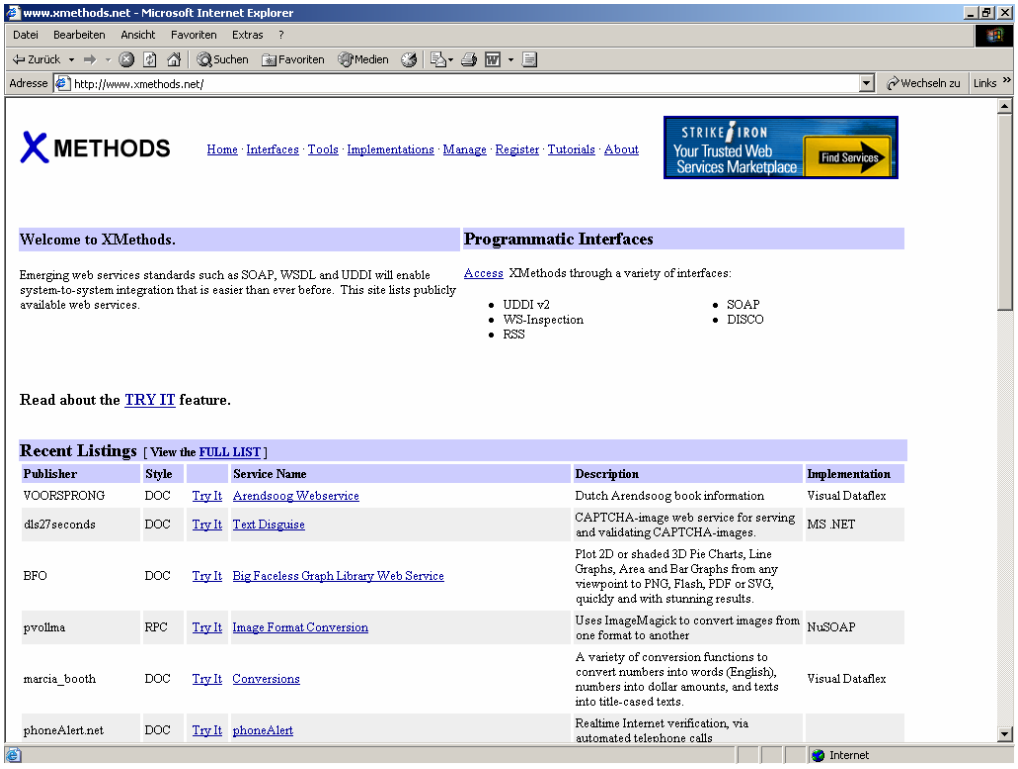

Abbildung 44: Verzeichnisdienst von xmethods

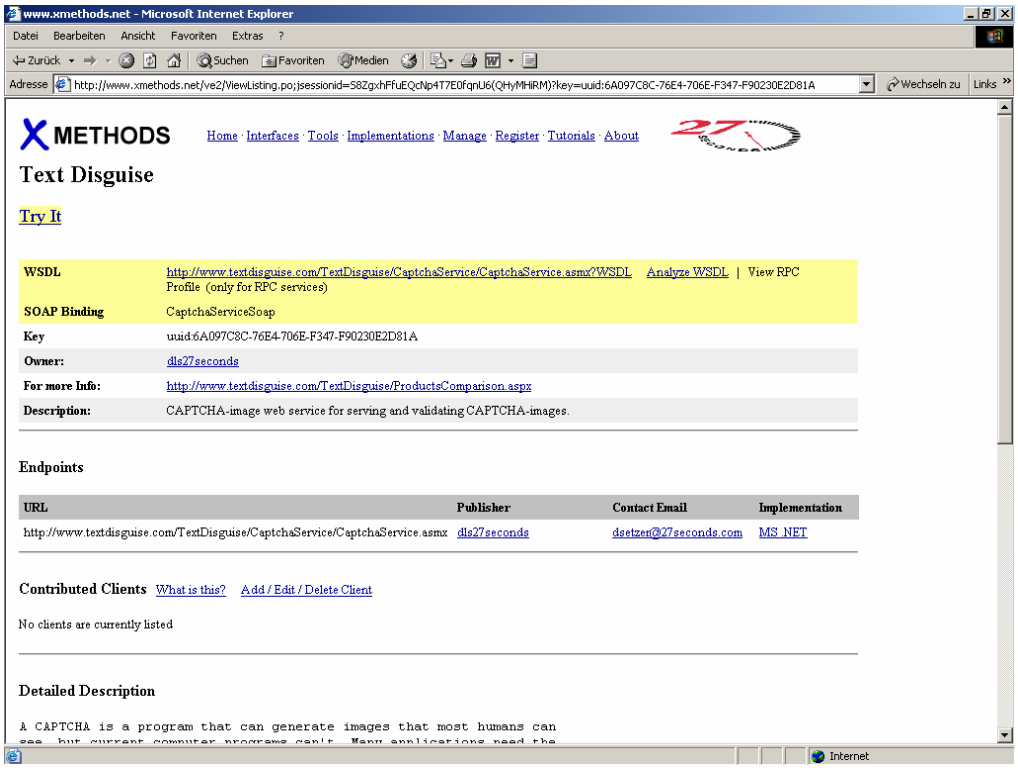

Abbildung 45: Informationen zum Service

Über den TryIt-Button gelangt der Nutzer zu einer ausgiebigen Testumgebung. Dabei handelt es sich um ein Diagnose-Werkzeug der Firma Mindreef (MINDREEF_2006), mit dem Web Services getestet, analysiert und verglichen werden können. Ebenso besteht die Möglichkeit, das WSDL-Dokument in verschiedenen Darstellungen (z.B. als Xml-Code, Baumansicht oder Graphik) einzusehen (Abbildung 46).

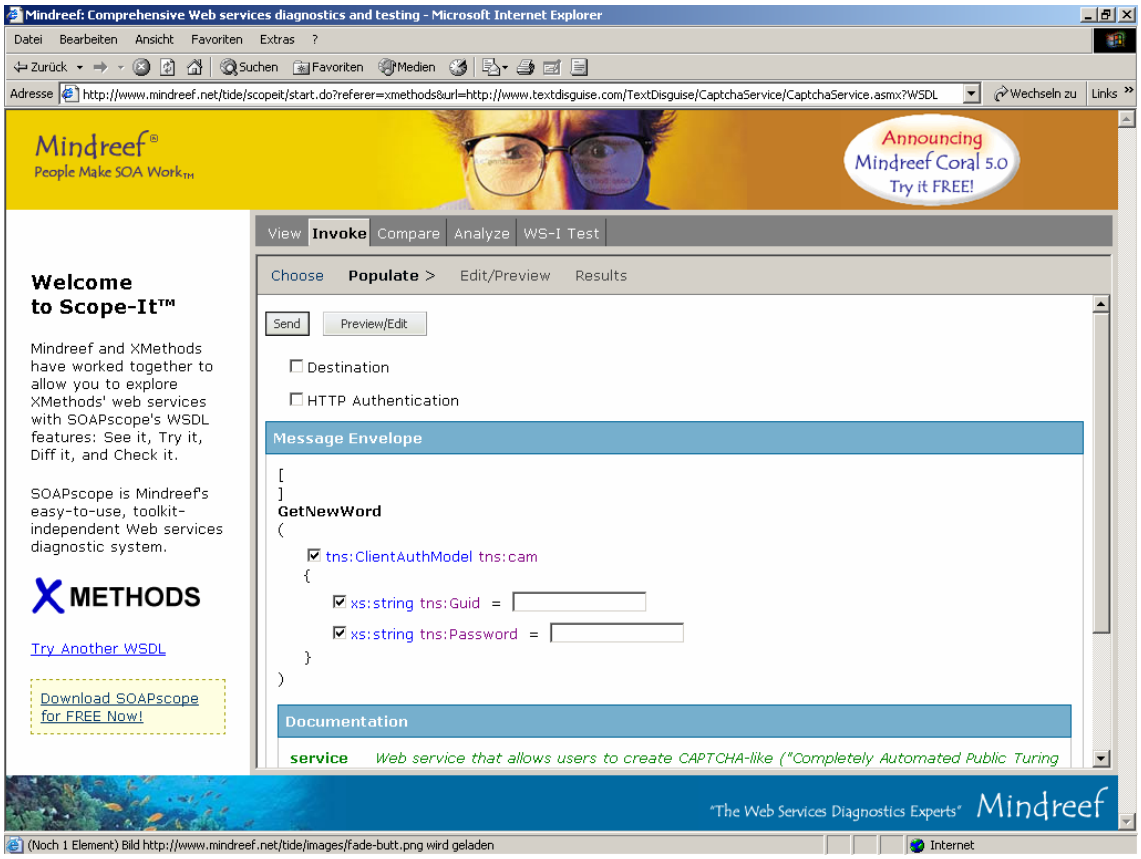

Abbildung 46: SOAPscope der Firma Mindreef

6.1.5 Zusammenfassung

Publikationsform	Bemerkung
Webseite	• relativ leicht zu erstellen • als Ergänzung für die manuelle Suche durch den Anwender.
WS-Inspection/DISCO	• im Vergleich zu UDDI einfach zu erstellen • ermöglicht automatisierte Suche • relativ geringer Aufwand für den Administrator, da Wartung auf dem eigenen Server • außerhalb des Intranet muß der URL des Einstiegsdokuments bekannt sein
UDDI	• sehr aufwendig und komplex, erfordert Einarbeitung • sinnvoll erst für eine große Anzahl an Diensten • automatisierte Suche möglich • standardisiert, breite Akzeptanz

Tabelle 2: Veröffentlichung von Web Services

6.2 Einsatz von Versionskontrollsystemen

Um die Weiterentwicklung des RipsFramework effizienter zu gestalten, bietet sich der Einsatz von Versionskontrollsystemen an, die den Entwickler bei der Erfassung und Rücknahme von Quellcode-Änderungen unterstützen. Die Programmierer können direkt auf die zentral abgelegten Daten zugreifen und Änderungen am Code vornehmen, ohne dass der Verlust älterer Versionen zu befürchten ist. Versionskontrollsysteme sind insbesondere bei der Koordination mehrerer Entwickler über einen längeren Zeitraum und bei räumlicher Trennung sehr hilfreich. Sie bieten grundsätzlich zwei Dienste an:

- *Dokumentationsfunktion:*

  Ermöglicht das Nachvollziehen von Quellcode-Änderungen. Dadurch kann der Zeitpunkt festgestellt werden, zu dem Fehler in die Anwendung eingeflossen sind.

- *Wiederherstellungsfunktion:*

  Ermöglicht die Rücknahme unerwünschter oder fehlerhafter Veränderungen.

Funktionsweise

Der Quellcode muss zunächst in das Versionsmanagement überführt und auf einem zentralen Server (Repository) abgespeichert werden. Dieses Repository besteht aus einem Dateibaum, der die Quelldateien in sämtlichen verfügbaren Versionen enthält und zusätzliche Informationen bereitstellt. Moderne Versionskontrollsysteme haben eine Client-Server-Architektur, d.h. der Entwickler kann sich über eine Client-Anwendung eine lokale Arbeitskopie der aktuellsten Version vom Server anfordern, mit der er wie gewohnt arbeiten kann (CheckOut). Die Rückführung der abgeänderten Codedateien (CheckIn) in das Repository erfolgt ebenfalls über ein spezielles Programm und unterliegt bestimmten Regeln, die zuvor vom Projektleiter festgelegt werden müssen. Eine Regel kann z.B. sein, dass der gespeicherte Quellcode kompilierbar und ausführbar sein muss, so dass die anderen Entwickler nicht in ihrer Arbeit behindert werden. Durch den Einsatz dieser Systeme können zudem verschiedene Konfliktsituationen behandelt werden, die bei der herkömmlichen Softwareentwicklung zu großen Problemen führten. Derartige Konflikte können auftreten, wenn zwei Programmierer zum gleichen Zeitpunkt Änderungen an einer identischen Datei vornehmen und sich über die Arbeit des anderen nicht bewusst sind. Sobald eine Person die geänderte Datei im Repository abspeichert, würden alle Veränderungen seit dem Öffnen der Datei überschrieben. Bei der Rückführung der lokalen Arbeitskopie in das Repository wird ein Abgleich zwischen dem aktuellen Stand auf dem Server und dem bearbeiteten Code durchgeführt. Dadurch bekommt der Entwickler aufgezeigt, was sich seit dem CheckOut geändert hat. Das Versionskontrollsystem ist in der Lage, diese Änderungen automatisch zusammenzuführen, sofern sich die Versionen nicht in der gleichen Datei und an der gleichen Stelle unterscheiden. In diesem Fall müssen die Entwickler manuell eingreifen (Budszuhn, 2005). Zusammengefasst bringt der Einsatz von Versionskontrollsystemen folgende Vorteile mit sich:

- *Durch die zentrale Speicherung in einem Repository ist auch ein zentrales Backup des Quellcodes möglich.*

- *Alte Versionsstände können problemlos wiederhergestellt werden.*

- *Die Entwickler haben über die Client-Programme immer Zugang zur aktuellsten Version der Quelldateien. Das funktioniert auch einwandfrei über das Internet.*

- *Die Zusammenführung verschiedener Änderungen ist relativ unproblematisch in der Handhabung.*

- *Versionskontrollsysteme ermöglichen durch Verzweigungen im Quellcode die parallele Entwicklung unterschiedlicher Versionen. Dadurch ist es möglich, neben der Entwicklung neuer Versionen die alten weiter zu pflegen und deren Fehler zu bereinigen.*

Derzeit gibt es auf dem Markt mehrere Systeme aus dem professionellen wie auch aus dem OpenSource-Bereich. Dazu gehören unter anderem Visual SourceSafe/Team Foundation Suite, Arch, Bitkeeper, Concurrent Versions System (CVS) und Subversion. Einige dieser Systeme sollen nachfolgend kurz vorgestellt werden.

6.2.1 Visual SourceSafe 2005 und Team Foundation Suite

Das neue Visual Studio 2005 ist als Standard-, Professional-, TeamSystem- oder in Form von mehreren Express-Versionen erhältlich. Mit Visual SourceSafe und dem Visual Studio Team Foundation Suite bietet Microsoft zwei Technologien für die Versionsverwaltung von Quellcode an (MSDN_2006). SourceSafe ist in der Visual Studio TeamSystem-Edition bereits integriert und für die Professional-Edition separat erhältlich. Dabei handelt es sich um ein relativ schlankes, projektorientiertes Versionskontrollsystem, das individuellen Programmierern oder kleineren Entwicklerteams Werkzeuge zur Verfügung stellt, mit denen Änderungen an Programmcode vorgenommen und projektübergreifend dokumentiert werden können. SourceSafe verwendet, wie die meisten anderen Systeme auch, ein CheckIn-/CheckOut-Modell und ist vorwiegend für die Verwaltung von Änderungen an Dateien und die Wiederherstellung alter Versionen konzipiert. Laut Microsoft (MSDN_2006) ist es am effektivsten für individuelle Programmierer oder kleinere Teams. Visual SourceSafe-Datenbanken können bis zu 4GB an Daten verwalten und sind über Visual Studio ansteuerbar. Der Vorteil von SourceSafe gegenüber anderen

Versionskontrollsystemen liegt in der optimalen Integration in die Visual Studio-Umgebung. Reichen die Funktionalitäten von SourceSafe nicht aus, kann auf die erweiterten Möglichkeiten der Team Foundation Suite zurückgegriffen werden. Dieses Paket stellt ein vollständiges Set an Werkzeugen für den gesamten Software-Entwicklungsprozess bereit.

6.2.2 Concurrent Versions System (CVS)

CVS ist eines der bekanntesten und am weitest verbreitenden Versionierungssysteme aus dem OpenSource-Bereich und wird seit 1986 entwickelt (CVS_2006). Es basiert auf dem Revision Control System (RCS), verfolgt aber einen neueren Ansatz bei der verteilten Entwicklung. RCS verwendet das Locking-Prinzip, bei dem eine Datei gesperrt wird, sobald ein Entwickler eine Kopie zur Bearbeitung angefordert hat. CVS dagegen ermöglicht die Verwendung mehrerer lokaler Arbeitskopien durch verschiedene Entwickler und führt die Änderungen beim CheckIn zusammen. Für den Zugriff auf den Server verwendet CVS standardmäßig eine Kommandozeilen-Applikation. Mit WinCvs oder gCvs für Linux sind aber auch eine Reihe grafischer Werkzeuge verfügbar. Neuerdings kann CVS als Add-In in verschiedene Entwicklungsumgebungen wie Eclipse und Microsoft Visual Studio eingebunden werden (Budszuhn, 2005).

6.2.3 Subversion

Subversion wird häufig als der Nachfolger von CVS angesehen, obwohl es eigentlich ein eigenständiges Projekt ist (TIGRIS_2006). Da es aber viele der Features von CVS übernommen hat und gleichzeitig dessen Fehler ausräumen möchte, kann durchaus von einer Neuauflage gesprochen werden. Einer der größten Nachteile von CVS ist, dass es keine Verzeichnisse versioniert und das Kopieren von Dateien nicht unterstützt. Subversion beseitigt unter anderem dieses Manko und beinhaltet zahlreiche zusätzliche Funktionen. Der Einsatz von Subversion hat sich zwar in den letzten Jahren bewährt, dennoch werden die meisten Open-Source-Projekte auf CVS-Basis entwickelt.

6.3 Quellcode-Verwaltung und Teamworking im RipsWeb

Der Quellcode der RipsWeb-Klassenbibliothek wird zunächst mit CVS verwaltet. Das CVS Repository mit der Masterversion des Frameworks befindet sich auf einem Server der Firma AHK in Freiburg und wird dort zentral administriert und gepflegt. Bei der LUBW in Karlsruhe wird eine mit einer Versionsnummer gekennzeichnete Arbeitskopie zentral abgelegt, auf die jeder Entwickler Zugriff hat. Dazu muss kein CVS installiert sein. Es ist allerdings wichtig, dass die CVS-relevanten Verzeichnisse und Dateien der Kopie nicht geändert werden. Die benötigten Drittbibliotheken wie CSharpZipLib oder log4net sowie die Dokumentation der Klassenbibliothek werden in separaten Verzeichnissen mitgeliefert. Die Microsoft .NET-Assemblies, das ESRI ArcGIS Framework und die Oracle-Bibliotheken (ODP.NET) müssen dagegen bei jedem Entwickler lokal installiert sein.

6.3.1 Lokale Erweiterung der Klassenbibliothek

Im Folgenden soll ein Weg aufgezeigt werden, wie die RipsWeb-Bibliothek bei der LUBW in weiterentwickelt werden kann. Zuerst besorgt sich der Entwickler die zentral abgelegte Arbeitskopie, mit welcher er lokal auf seinem eigenen Rechner arbeitet. Die aktuellste Version befindet sich derzeit auf dem Server \\itzgs2\entwicklung. Auf dem lokalen Arbeitsplatz wird diese Kopie in einem entsprechenden Verzeichnis abgelegt (z.B. C:\RipsFramework). Nach dem Kopieren muss der Schreibschutz entfernt werden, um die Solution in einem späteren Schritt kompilieren zu können. Anschließend kann sich der Entwickler seine eigene Umgebung einrichten. Die Abbildung 47 zeigt einen unverbindlichen Vorschlag, wie diese Umgebung aussehen könnte. Im Verzeichnis RipsFramework wird neben der Arbeitskopie auch ein Unterverzeichnis „MyRipsWeb“ eingerichtet, in dem eigene Projekte abgelegt werden können. Zudem

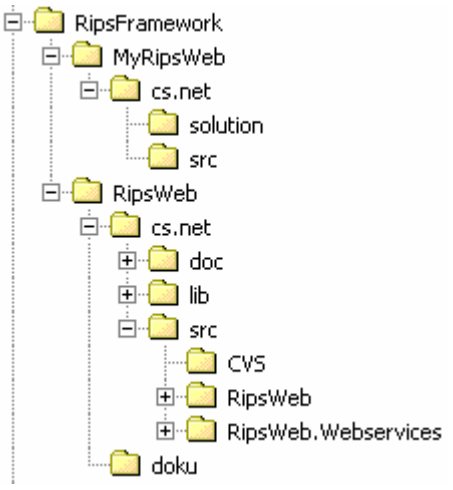

Abbildung 47: Lokale Arbeitskopie des RipsWeb

können hier auch Test-Clients und Applikationen gespeichert werden, die RipsWeb nutzen, aber nicht in das Framework integriert werden sollen.
Im Verzeichnis MyRipsWeb kann nun eine Solution-Datei erstellt werden, welche die RipsWeb-relevanten Projekte einbindet und innerhalb derer neue Web Services, Klassen und Client-Applikationen programmiert werden. Dazu wird im Solution-Verzeichnis eine leere Projektmappe erzeugt und dieser das RipsWeb-Projekt aus der Arbeitskopie hinzugefügt. Im nächsten Schritt müssen die Verweispfade an die lokalen Bedingungen angepasst werden (Abbildung 48). Das geschieht am einfachsten, indem in den Eigenschaften des Teilprojektes die entsprechenden Pfade einer Liste hinzugefügt werden. Diese Liste wird durchsucht, sobald das Projekt für die Assemblies geladen wird (Menü „Projekt“ → „Eigenschaften“ → „Verweispfad“). Dabei ist zu beachten, dass Verweise zu den Drittbibliotheken auf das lib-Verzeichnis der lokalen Arbeitskopie gerichtet sind. Die Verweise zu den lokal installierten Bibliotheken des .NET-Frameworks, der ArcObjects und der ODP.NET werden dagegen über den Global Assembly Cache (GAC) aufgelöst und müssen nicht in der Liste referenziert werden.

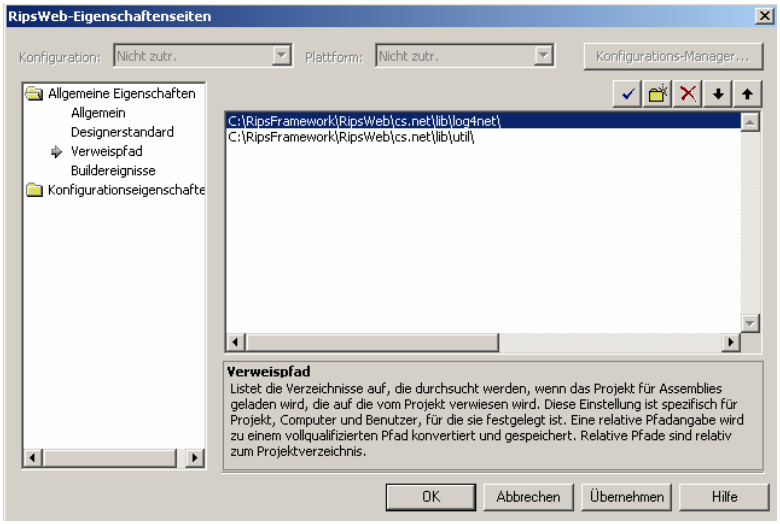

Abbildung 48: Anpassen der Verweispfade

Anschließend kann das RipsWeb-Projekt kompiliert werden. Beim Build-Vorgang wird neben dem dll-File auch die Datei *RipsWeb.csproj.user* erzeugt, in der unter anderem die *ReferencePath*-Informationen gespeichert werden. Es ist sinnvoll, diese Datei nach dem erstmaligen Einrichten der lokalen Umgebung separat zu sichern, um sie für die

Arbeit mit zukünftigen Versionen des CVS Repository wieder verwenden zu können (Abbildung 49). Dies kann zum Beispiel in „MyRipsWeb“ erfolgen, in dem analog zur Arbeitskopie die RipsWeb-Verzeichnisse zur Speicherung der nutzerspezifischen Dateien angelegt werden. Wird zu einem späteren Zeitpunkt eine neue Arbeitskopie bezogen, können diese Dateien einfach überschrieben werden. Auf diese Weise müssen die Verweispfade nur einmal an die lokalen Bedingungen angepasst werden.

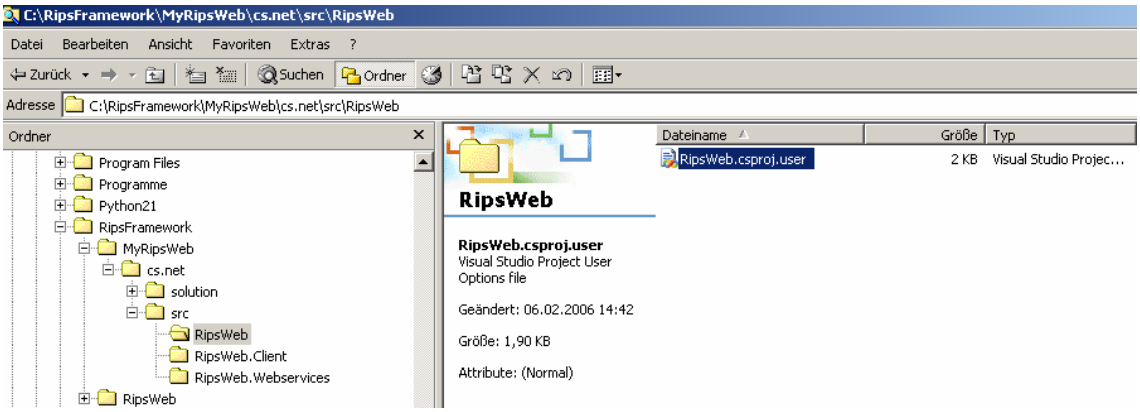

Abbildung 49: Beispiel für die Verwaltung lokaler nutzerrelevanter Projekteigenschaften

Um der Mappe das RipsWeb.Webservices-Projekt hinzuzufügen, muss in der Computerverwaltung zunächst ein virtuelles Verzeichnis auf dem Web Server (IIS) eingerichtet werden, dessen Pfad auf das Web Service-Projekt der Arbeitskopie zeigt (Abbildung 50). Idealerweise wird der gleiche Alias wie jener der Arbeitskopie (RipsWeb) verwendet. Soll eine andere Bezeichnung vergeben werden, ist der Pfad in der Webinfo-Datei des Web Service-Projektes anzupassen.

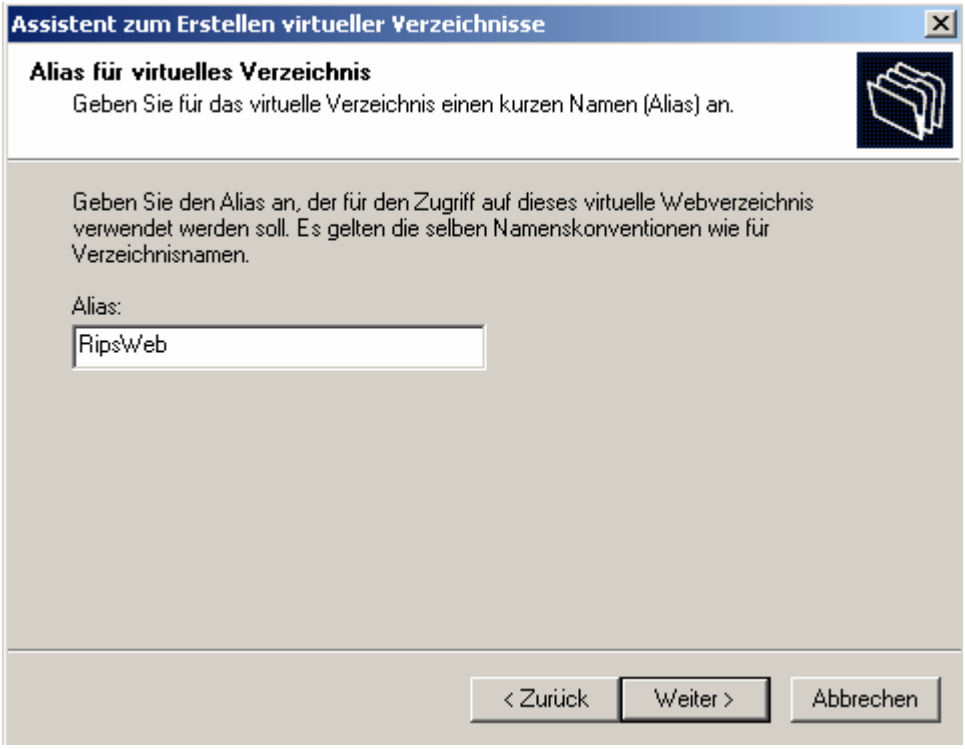

Abbildung 50: Einrichten eines virtuellen Verzeichnisses

In den Eigenschaften des virtuellen Verzeichnisses wird im Register „Verzeichnissicherheit“ das Benutzerkonto des Entwicklers authentifiziert. An dieser Stelle sollte überprüft werden, ob dieser auch über alle für sein Projekt relevanten Zugriffsrechte verfügt (z.B. für den ArcGIS Server, Oracle Spatial oder ArcSDE). Nach der Einrichtung des virtuellen Verzeichnisses empfiehlt es sich, die Standardwebsite zu beenden und wieder neu zu starten. Nun kann das RipsWeb.Webservices-Projekt der Solution hinzugefügt werden (Abbildung 51). Auch hier sollten nochmals die Verweispfade überprüft werden.

Da RipsWeb.Webservices die RipsWeb-Assembly einbindet, muss dafür gesorgt werden, dass RipsWeb vor dem Web Service-Projekt kompiliert

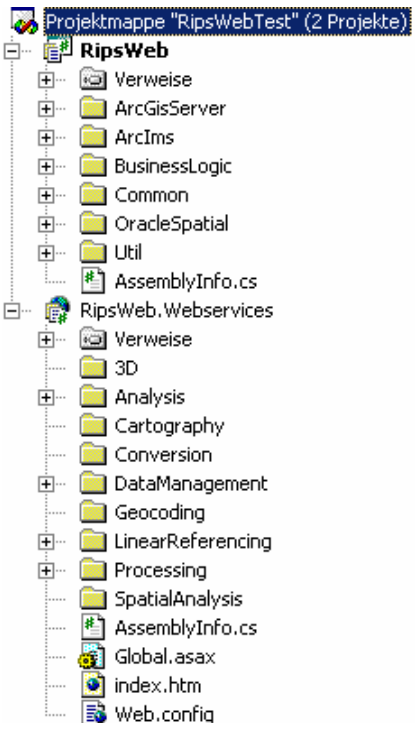

Abbildung 51: Hinzufügen von RipsWeb.Webservices

wird. Dazu sind für die Projektmappe Abhängigkeiten zu definieren (Abbildung 52):

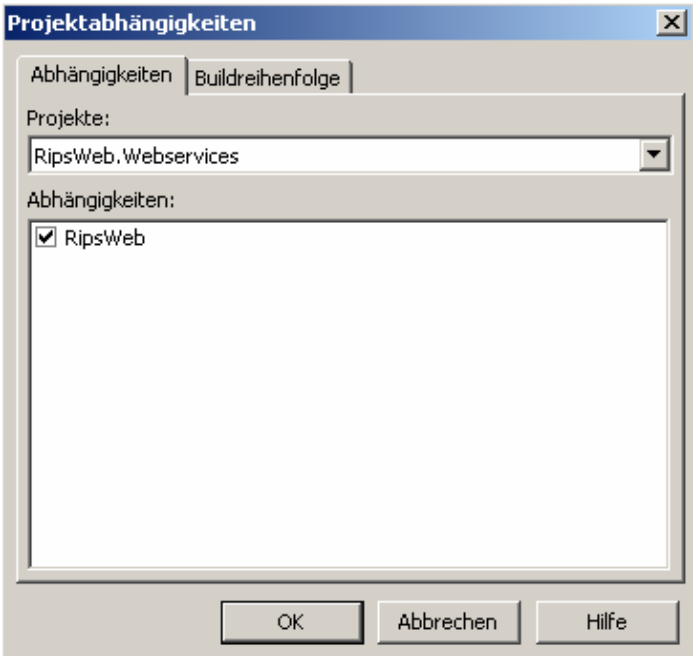

Abbildung 52: Festlegung von Projektabhängigkeiten

Die Build-Reihenfolge sieht schließlich folgendermaßen aus:

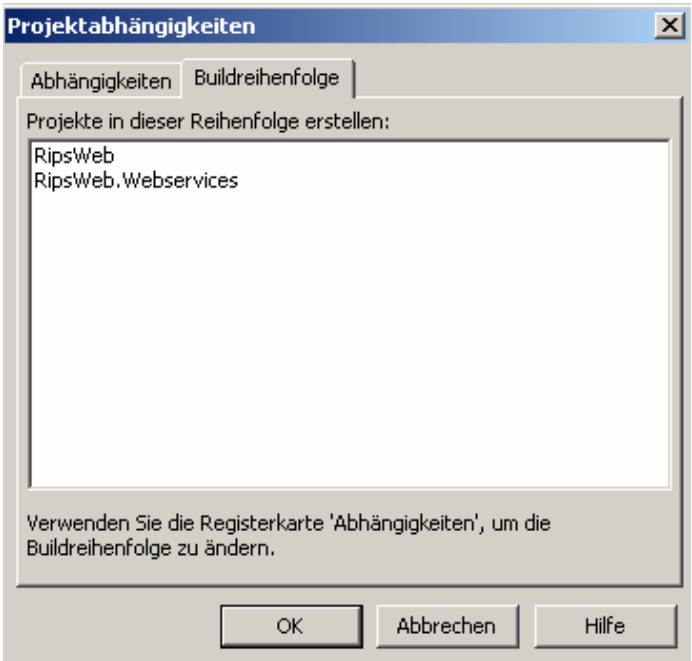

Abbildung 53: Buildreihenfolge

Als nächstes müssen die Parameter für den Datenzugriff an die lokalen Bedingungen angepasst werden. Diese werden aus der Web Service-Konfigurationsdatei (bzw. aus der Anwendungskonfigurationsdatei des Client) ausgelesen.

```
<configuration>
  <appSettings>
    <add key="PGDB" value="\\..."></add>
    <add key="RASTERFILE_PATH" value="\\..."></add>
    <add key="SHAPEFILE_PATH" value="\\..."></add>

    <add key="ORACLE_SERVER" value=…"></add>

    <add key="SDE_SERVER" value=""></add>
    <add key="SDE_INSTANCE" value=""></add>
    <add key="SDE_DB" value=""></add>
    <add key="SDE_USER" value=""></add>
    <add key="SDE_PWD" value=""></add>
    <add key="SDE_VERSION" value=""></add>
  </appSettings>
```

Im letzten Schritt wird die Projektmappe kompiliert. Nun können die Web Services mit Hilfe der ASP.NET-Benutzerschnittstelle getestet werden. Bei Zugriff über den ArcGIS Server muss die obige *Web.config* zusätzlich mit dessen Parameter ergänzt werden. Die verfügbaren Web Services können jetzt in eigene Projekte eingebunden, und das Framework um zusätzliche Methoden und Klassen erweitert werden (Abbildung 54). Dabei erfolgt die Modifizierung direkt im Quellcode der lokalen Arbeitskopie.

Ist die Projektarbeit abgeschlossen, wird die lokale Arbeitskopie mit den Quellcode-Änderungen als gepackte Datei zur Übernahme in das CVS Repository zur AHK nach Freiburg geschickt. Dort können sie mit der Version auf dem Server synchronisiert und eingecheckt werden. Hierfür sind die in der Arbeitskopie enthaltenen CVS-Informationen wichtig, die keinesfalls geändert werden dürfen.

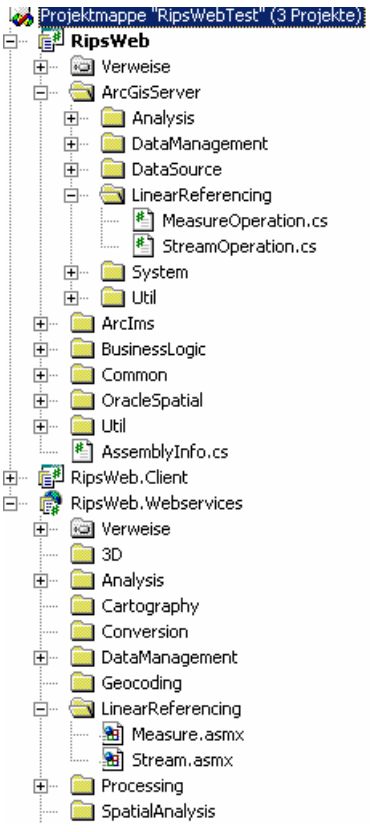

Abbildung 54: Hinzufügen neuer Klassen

Die gepackte Datei sollte nach folgendem Muster benannt werden:

`[Entwickler]_[Projektname].zip`, z.B. `53DSH_Measure.zip`

Es empfiehlt sich, bei der Einrichtung der lokalen Umgebung auch die Web.config sowie die Webinfo-Datei des Web Service-Projektes separat zu sichern. Bei der nächsten Version der Arbeitskopie können diese Dateien dann einfach überschrieben werden, so dass die Verbindungsparameter zur Datenquelle und der Pfad zum virtuellen Verzeichnis auf dem IIS nicht erneut angepasst werden müssen (Abbildung 55).

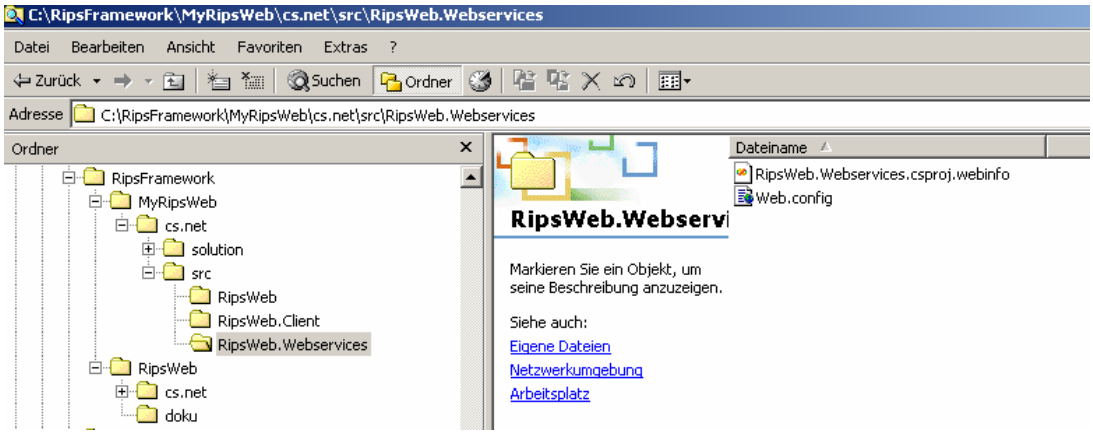

Abbildung 55: Speicherung der nutzerspezifischen Informationen für das RipsWeb.Webservices-Projekt

Ist die eigene Umgebung einmal eingerichtet, muss der Entwickler nach Erhalt einer neuen Version aus dem Repository nur noch folgende Schritte durchführen:

- *Entfernen der alten Arbeitskopie (dazu muss evtl. der aspnet_wp.exe-Prozess im TaskManager beendet werden)*
- *Kopieren der neuen Arbeitskopie auf den lokalen Rechner*
- *Entfernen des Schreibschutzes*
- *Erzeugen der RipsWeb.dll*
- *Überschreiben der RipsWeb.csproj.user sowie der Webinfo- und Web.config-Dateien der Arbeitskopie mit den nutzerspezifischen Versionen*

6.3.2 Coding Conventions für RipsWeb

Um die Dokumentation (Abbildung 56) der RipsWeb-Klassenbibliothek zu erstellen, müssen neu angelegte C#-Dateien folgende Kommentierungen enthalten:

Header

Zu Beginn der Datei wird ein Header eingefügt. Die beiden Parameter Revision und Date werden beim CheckIn in das CVS Repository aktualisiert und ersetzt:

```
// RipsWeb Framework
// Copyright (C) 2006, LUBW - Landesanstalt für Umwelt, Messungen
und Naturschutz Baden-Württemberg
//
// $Revision: 1.4 $
// $Date: 2006/01/30 16:41:17 $
//
```

Kommentierung der Klasse

Die Kommentierung der jeweiligen Klasse muss eine kurze Beschreibung der Funktionalität, den Autor und dessen aktuelle eMail-Adresse enthalten:

```
/// <summary>
/// Zusammenfassung für MeasureOperation.
/// <br>
/// Autor: <a href="mailto:stefan.haberer@lubw.bwl.de">Stefan
Haberer</a>
/// </summary>
```

Kommentierung der Membervariablen, Konstruktoren und Methoden

```
/// <summary>
/// Host name where spatial operations take place
/// <summary>
private string m_strHost;


/// <summary>
/// constructor
/// <summary>
///<param name="p_strHost">Host name</param>
private ArcGisOverlay(string p_strHost)
{
 m_strHost = p_strHost;
}
```

```
/// <summary>
/// Get feature count as result of spatial operation between two
    feature classes.
/// </summary>
/// <param name="p_strDataSource">defined ArcGISServer Server
      Object</param>
/// <param name="p_strSrcFc">name of source feature
      class</param>
/// <param name="p_strSrcFcWhereStmt">where statement for query
      on source feature class</param>
/// <param name="p_iSpatialRelation">spatial operation:
      intersect, contain, are contained by ...</param>
/// <param name="p_strTargetFc">name of target feature
      class</param>
/// <returns>The number of features of the target feature class
      hit by the spatial operation</returns>
public int getFeatureCount(string p_strDataSource, string
  p_strSrcFc, string p_strSrcFcWhereStmt, int
  p_iSpatialRelation, string p_strTargetFc)
{
}
```

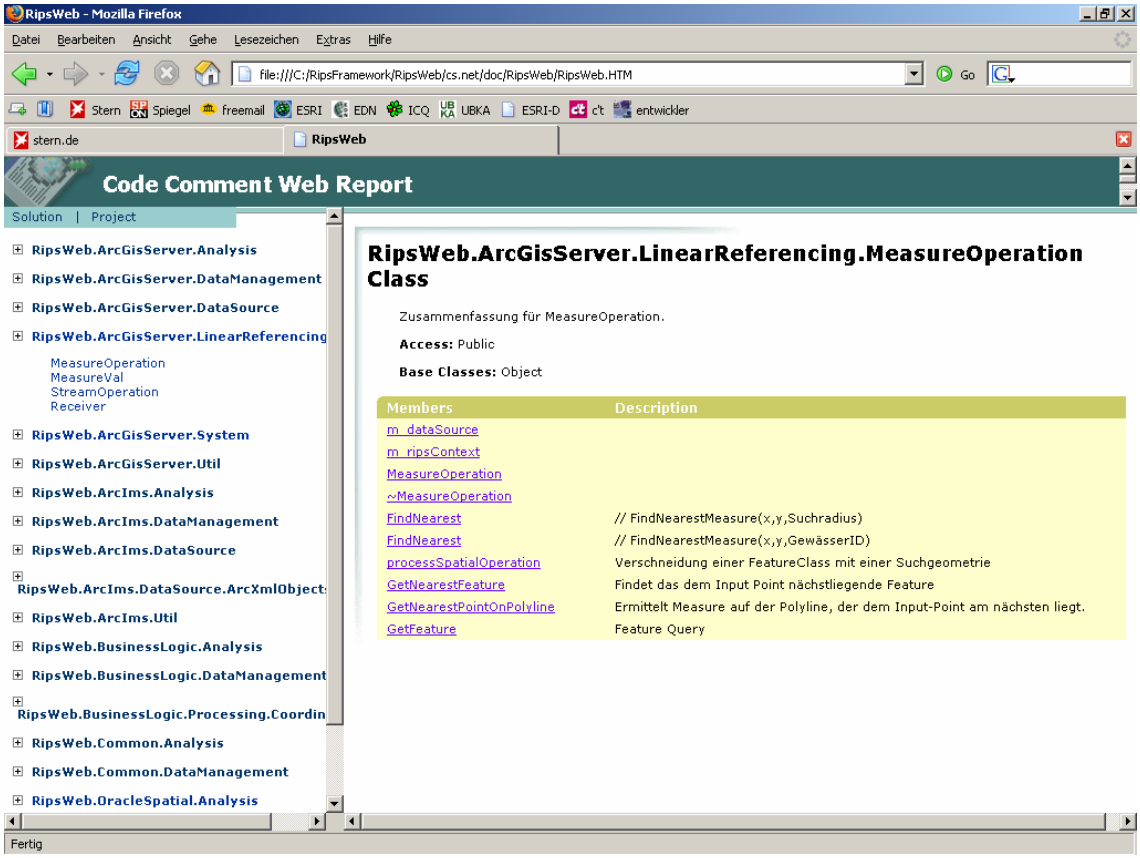

Abbildung 56: Dokumentation der Klassenbibliothek

7 Performance Tuning

7.1 Erweiterung des GIS Servers

Wie bereits erwähnt wurde, verwendet der Server die gleichen ArcObjects wie das Desktop GIS und ArcGIS Engine. Prinzipiell gilt, dass ArcObjects-basierte GIS-Funktionalitäten, die in diesen Umgebungen nicht leistungsfähig sind, auch nicht in der Server-Umgebung performant sein können. Im Unterschied zur Desktop-Ausführung erfolgen ArcObjects-Aufrufe über mehrere Prozesse und meist über mehrere Rechner. So befindet sich die aufrufende Web-Applikation in Prozessen auf dem Web Server, während das Objekt auf einem SOC-Rechner ausgeführt wird. Bei einer geringen Anzahl von Aufrufen hat dies noch keine Bedeutung für die Geschwindigkeit einer Anwendung. Steigt die Zahl jedoch, kann das enorme Auswirkungen auf die Performance haben. Daher ist es wichtig, die Aufrufe von feinkörnigen Objekten auf ein Minimum zu reduzieren. Ist das nicht möglich, können spezielle COM-Objekte erzeugt werden, die direkt auf den SOC-Rechnern installiert werden, und dort die Arbeit für die Web-Applikation verrichten. Dadurch können jene Funktionalitäten, die häufig feinkörnige Objekte aufrufen, Server-seitig verlagert werden. Der ArcGIS Server Administrator and Developer Guide (ESRI, 2004) enthält ausführliche Beispiele für die Erstellung dieser Objekte.

7.2 Beschränkung der Ergebnismenge

Auch die Anzahl der zurückgelieferten Ergebniswerte kann das System schnell ausbremsen. Der Entwickler sollte darauf achten, dass der Anwender keine Abfragen ausführen kann, die den Web Server bzw. die Datenbank so stark beanspruchen, dass diese nicht mehr für die Nutzung durch andere zur Verfügung stehen. Das umfasst zum einen die Überprüfung der übergebenen Parameter und zum anderen die Einschränkung der Anzahl an zurückgelieferten Ergebnisse.
Schreibt eine Applikation Daten in das Server Directory, muss bedacht werden, dass große Dateien Rechenleistung und Speicher benötigen. Die Größe eines von der *ExportMapImage*-Methode des MapServer Objects erzeugten Bildes kann im MapServer-Configuration File begrenzt werden. Auch der Output von GeocodeServer

Objects kann eingeschränkt werden. Diese Parameter müssen mit dem GIS Server Administrator abgestimmt werden.

7.3 Verwendung von pooled Server Objects

Werden für die Durchführung von GIS-Operationen keine festen MapServer- oder GeocodeServer Objects verwendet, besteht die Möglichkeit, mit der *CreateObject*-Methode Objekte innerhalb leerer ServerContexts zu erzeugen. Die Objekterzeugung kann jedoch in manchen Fällen sehr viele Ressourcen beanspruchen. Ein Beispiel ist der Aufbau einer Verbindung zu einem Geodatabase-Workspace. Wird bei jedem Aufruf eines Web Service eine neue Verbindung aufgebaut, so wirkt sich das auf die Leistungsfähigkeit der Applikation und des Datenbank Servers aus. Daher sollte bei Services, die eine Verbindung zur Datenbank benötigen und häufig aufgerufen werden, ein Objekt-Pool verwendet werden. Dieser Pool baut einmalig eine bestimmte Anzahl an Workspace-Verbindungen auf, die unter den Klienten aufgeteilt werden. Technisch kann dies über ein Map-Dokument gelöst werden, das einen Layer mit der FeatureClass des Workspace enthält. Aus dieser mxd-Datei wird ein pooled MapServer Object generiert, dessen Objektinstanzen jeweils eine Verbindung zum Workspace halten. Wird ein Web Service aufgerufen, erhält er eine Referenz auf eine MapServer-Instanz und gibt diese an den Objekt-Pool zurück, sobald er den Workspace nicht mehr benötigt.

7.4 ASP.NET Caching

Der Cache ermöglicht die Ablage von Informationen im Arbeitsspeicher, um zukünftige Anfragen schneller bearbeiten zu können. Der Vorteil des Caching besteht darin, dass die erneute Erzeugung von Daten meist wesentlicher aufwendiger ist, als das Auslesen aus dem Cache. Die Wiederverwendung von Informationen spart Ressourcen ein, entlastet den Web Server und kann dadurch die Antwortzeiten für den Client enorm reduzieren. Caching wird vorwiegend eingesetzt, um so genannte „Round Trips“ zum Server und Zugriffe auf die Datenquelle zu minimieren, da diese Vorgänge Web Services massiv verlangsamen können. Das Caching mit ASP.NET ist sehr komfortabel, da das System die gesamte Verwaltung übernimmt. Nicht benötigte

Elemente werden automatisch aus dem Speicher entfernt, sobald ihre Gültigkeit abgelaufen ist oder die Speicherressourcen knapp werden. Nach Ferrara&McDonald (2002) gibt es zwei Arten von Caching, die von ASP.NET unterstützt werden:

- *Output caching*
- *Data caching*

Die beiden Typen sind komplementär und können parallel in einer Web-Methode verwendet werden.

7.4.1 Output caching

Beim Output caching wird nur das Ergebnis einer Web-Methode im Cache abgelegt. Wird der Dienst mit denselben Parametern innerhalb eines bestimmten Zeitraumes noch mal aufgerufen, kann der Wert aus dem Cache ausgelesen werden. Diese Art des Caching erfolgt weitestgehend automatisch und ist einfach zu implementieren, da kaum zusätzlicher Code benötigt wird (Abbildung 57).

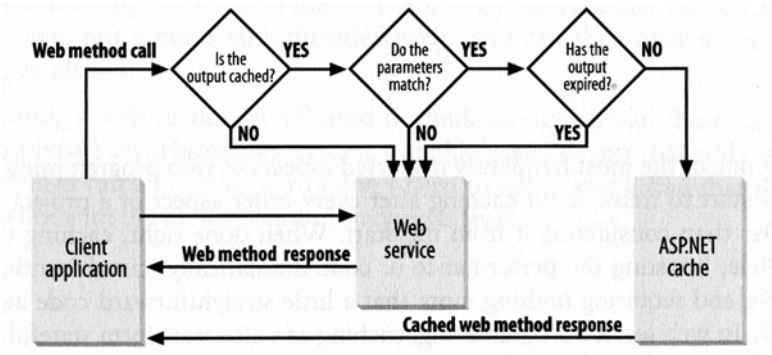

Abbildung 57: Funktionsweise von Output caching

Um Output caching zu aktivieren, muss lediglich die *CacheDuration*-Eigenschaft des *WebMethod*-Attributes festgelegt werden. Dieser Wert gibt die Anzahl der Sekunden an, die das Ergebnis im Speicher gehalten werden soll:

```
[WebMethod(CacheDuration=60)]
```

Output caching ist aber nicht für alle Web-Methoden geeignet, da es den Arbeitsspeicher stark belasten kann. Der Einsatz sollte für jede Funktion gesondert überlegt werden. Ein wichtiger Aspekt ist die Aktualität der Cache-Daten. Je nachdem, wie lange die Ergebnisse gehalten werden, muss sich der Entwickler die Frage stellen, ob die Gültigkeit der Daten noch gegeben ist.
Grundsätzlich sollte Output caching immer dann in Erwägung gezogen werden, wenn der Web Service Operationen mit zeitintensiven Datei- oder Datenbankzugriffen ausführt. Dabei sollte der Dienst aber nicht allzu viele unterschiedliche Ergebnisse liefern können. Wird für jede Anfrage ein anderer Wert ermittelt, macht die Anwendung von Output Caching keinen Sinn. Der Speicher würde sehr schnell überlastet, während der Nutzen des Caching gering wäre. Output caching sollte auf keinen Fall eingesetzt werden, wenn Web-Methoden Abhängigkeiten zu anderen Objekten haben und diese zusätzlich zu den Übergabeparametern das Ergebnis beeinflussen können.
Das Output caching ist standardmäßig durch Hinzufügen der *CacheDuration*-Eigenschaft aktiviert. Soll es für alle Applikationen deaktiviert sein (z.B. in der Entwicklungsphase), kann der *OutputCache*-Eintrag in *<httpModules>* auskommentiert werden. Der Eintrag befindet sich in der machine.config-Datei unter *C:\[WindowsDir]\Microsoft.NET\[version]\Config*:

```
<httpModules>
<!--
  <add name="OutputCache"
       type="System.Web.Caching.OutputCacheModule"/>
-->
</httpModules>
```

Alternativ kann das Output caching für eine einzelne Applikation deaktiviert werden. Das OutputCache-Modul kann entfernt werden, indem der web.config-Datei folgender Code hinzugefügt wird:

```
<httpModules>
  <remove name="OutputCache"/>
</httpModules>
```

7.4.2 Data caching

Während Output caching nur das Ergebnis einer Web-Methode zwischenspeichert, kann Data caching (Abbildung 58) alle Arten von Daten im Cache ablegen. Es bietet außerdem die Möglichkeit, auf die Lebensdauer eines Caching-Objektes und dessen Abhängigkeiten Einfluss zu nehmen. Allerdings ist die Verwendung von Data caching im Vergleich zum Output caching wesentlich aufwendiger, da zusätzlicher Code in der Web-Methode benötigt wird. Bei der Programmierung sollte darauf geachtet werden, dass die Caching-Funktionalität von der übrigen Logik getrennt wird.

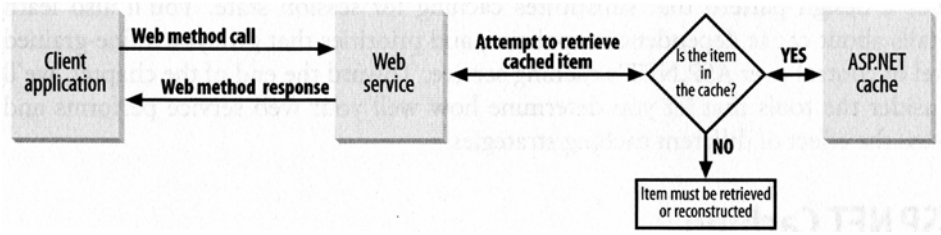

Abbildung 58: Funktionsweise von Data caching

Die Basis für das Data caching bildet die *System.Web.Caching.Cache*-Klasse, die den Cache für eine Web-Anwendung implementiert. Eine Instanz dieser Klasse liefert eine *dictionnary collection* zurück, über die auf jedes Element im Cache zugegriffen werden kann. Die einzelnen Elemente werden über einen Schlüssel indexiert. Um auf die Cache collection eines Web Service zuzugreifen, wird die *Context.Cache*-Eigenschaft verwendet, sofern die Web Service-Klasse von *System.Web.Services.WebService* abgeleitet ist. Andernfalls erfolgt der Zugriff über *HttpContent.Current.Cache*. Die *Insert*-Funktion dient dazu, dem Cache Elemente unter der Verwendung eines Schlüssels hinzuzufügen:

```
Context.Cache.Insert(key, value);
```

Weiter existieren mehrere Überladungen, mit denen sämtliche Aspekte der „Cache policy“ selbst festgelegt werden können:

```
Context.Cache.Insert(key, value, dependencies);
Context.Cache.Insert(key, value, dependencies,
absoluteExpiration, slidingExpiration);
Context.Cache.Insert(key, value, dependencies,
absoluteExpiration, slidingExpiration, priority,
onRemoveCallback);
```

AbsoluteExpiration, SlidingExpiration und Priority

Beim Data caching kann die Lebensdauer von Cache-Elementen mit zwei Parametern bestimmt werden. Mit *absoluteExpiration* wird eine Zeitspanne festgelegt, für welche die Daten im Cache verbleiben sollen. Das ist sinnvoll, wenn bekannt ist, dass ein Datensatz nur für eine bestimmte Zeit gültig ist und dann erneuert werden muss. *SlidingExpiration* bedeutet, dass ein Cache-Element gelöscht wird, sobald es innerhalb eines bestimmten Zeitraumes nicht abgefragt wurde. Dieser Parameter wird gesetzt, wenn sich die Daten nicht verändern und nur selten abgerufen werden. Darüber hinaus können je nach *Insert*-Überladung Cache-Elementen Prioritäten vergeben werden. Sobald ASP.NET feststellt, dass der Speicher knapp wird, werden automatisch Objekte aus dem Cache entfernt. Durch die Zuweisung von Prioritäten kann festgelegt werden, welche Daten zuerst gelöscht werden sollen. Die weiteren Parameter der *Insert*-Funktion sind:

- *Dependencies*

  Das sind die Dateien oder Verzeichnisse, von denen die gecachten Daten abhängig sind. Werden diese Dateien geändert, löscht ASP.NET die betroffenen Daten aus dem Cache.

- *onRemoveCallback*

  Ein Delegat, der aufgerufen wird, wenn Daten aus dem Cache entfernt werden. In der Regel wird es dazu verwendet, die Abhängigkeiten der gelöschten Daten zu entfernen, neue Daten in den Cache zu laden oder Diagnose-Funktionen auszuführen.

7.4.3 Beispiel Data caching

Das folgende Listing zeigt ein Anwendungsbeispiel für das Data caching. Die Web-Methode *getFeatureClassNames* ermittelt alle FeatureClasses einer Datenquelle und liefert eine Liste mit deren Namen zurück. Die Funktion prüft zunächst, ob sich die Liste aus vorhergehenden Abfragen noch im Cache befindet. Ist dies nicht der Fall, wird sie mit *getFeatureClassNamesList* neu erstellt und für eine Dauer von 60 Sekunden zwischengespeichert.

```
[WebMethod]
public ArrayListgetFeatureClassNames(Enumerations.WorkspaceType
  p_wsType)
{
  // Check cache for the ArrayList. If it ist found, retrieve
  // it. If not, create it by calling getFeatureClassNamesList
  // and add it to the cache
  ArrayList l_listRet = new ArrayList();

  if (Context.Cache["FeatureClassNamesList"] == null)
  {
    l_listRet = getFeatureClassNamesList(p_wsType);
    // Set time stamp
    l_listRet.Add(System.DateTime.Now.ToString());
    Context.Cache.Insert("FeatureClassNamesList", l_listRet,
      null, DateTime.Now.AddSeconds(60), TimeSpan.Zero);
  }
  else
  {
    l_listRet =
      (ArrayList)Context.Cache["FeatureClassNamesList"];
  }

  return l_listRet;
}

private ArrayList getFeatureClassNamesList(
  Enumerations.WorkspaceType p_wsType)
{
  ArrayList l_listRet = new ArrayList();

  try
  {
    FeatureDataset l_md = new FeatureDataset(p_wsType);
    l_listRet = l_md.getDatasetNames();
  }
  catch(Exception ex)
  {
    SoapException se = create(ex, Context);
    throw se;
```

```
        }

        return l_listRet;
      }
```

Der Zeitstempel dient der Überprüfung, ob tatsächlich die gecachten Daten verwendet werden. Innerhalb des Fensters von 60 Sekunden wird für jede Anfrage der gleiche Zeitpunkt angegeben (Abbildung 59). Danach wird das Objekt aus dem Cache gelöscht und bei erneutem Ausführen der Web-Methode wieder aufgebaut. Der Geschwindigkeitszuwachs bei Verwendung der gecachten Daten ist dabei deutlich erkennbar.

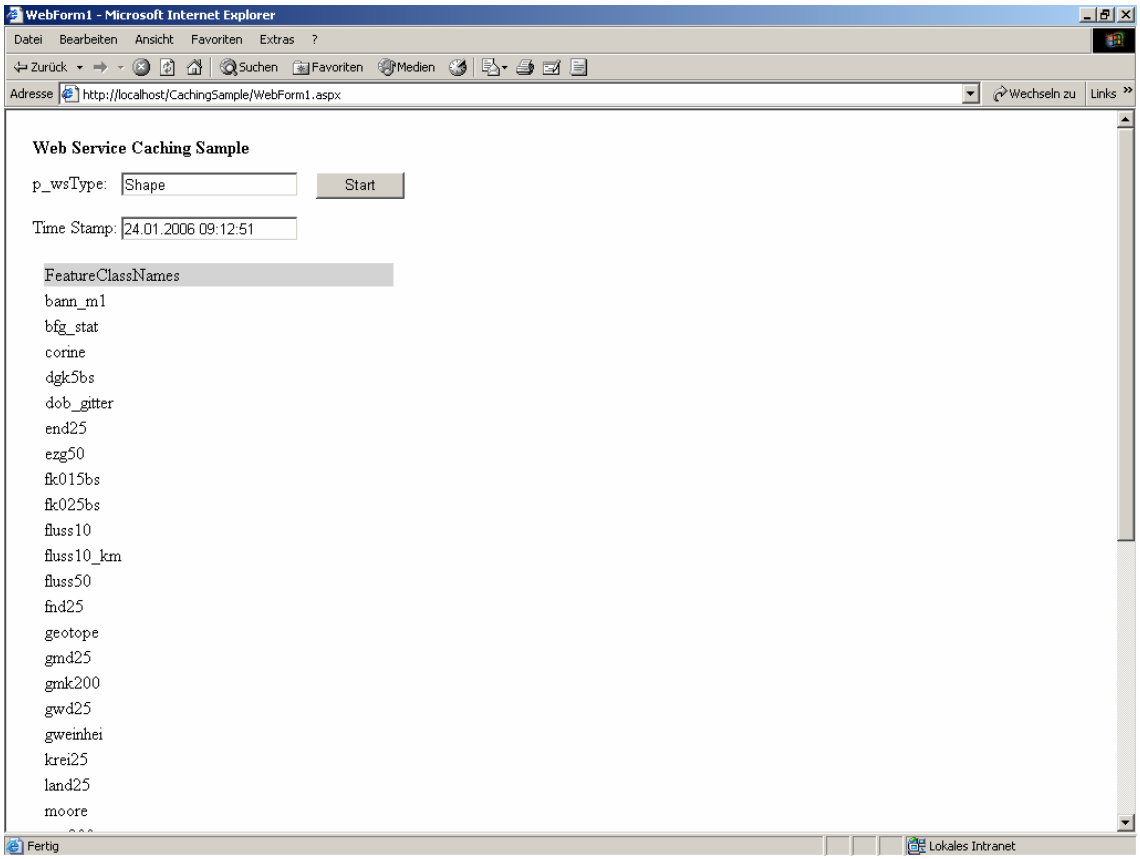

Abbildung 59: Beispiel für Data caching

7.4.4 Caching Application Block der Microsoft Enterprise Library

Die Microsoft Enterprise Library beinhaltet eine Sammlung so genannter „Application Blocks", die den Programmierer bei der Erstellung verteilter Anwendungen unterstützen sollen. Die einzelnen Application Blocks stellen Standardfunktionalitäten aus verschiedenen Bereichen bereit, die entweder unverändert eingebunden oder vom Entwickler angepasst werden können. Microsoft bezeichnet sie als „recommended best practices for .NET applications" (MSDN_2006). Die Enterprise Library bietet Application Blocks für die Bereiche „Data Access", „Configuration", „Cryptography", „Exception Handling", „Logging and Instrumentation", „Security" und „Caching". So erweitert der Caching Application Block das ASP.NET Caching um zusätzliche Funktionen und unterstützt auch das Caching für andere Anwendungen wie Windows Forms, Konsolenanwendungen und Windows-Diensten. Weitere Informationen zur Enterprise Library und zum Caching Application Block finden sich auf der Webseite von Microsoft (MSDN_2006) sowie auf den Seiten des Entwickler-Magazin (ENTWICKLER_2006).

8 Zusammenfassung und Ausblick

In der Einführung dieser Diplomarbeit wurden die verschiedenen GIS-Werkzeuge und deren Einsatzbereiche vorgestellt, die im Rahmen des Räumlichen Informations- und Planungssystems (RIPS) bzw. dem Umweltinformationssystem des Landes Baden-Württemberg verwendet werden.

Ziel des RipsFramework ist die Zusammenführung der Funktionalitäten dieser Werkzeuge auf einer einheitlichen Basis (.NET). Dabei sollen getrennt voneinander ein Desktop- und ein Webservice Framework entwickelt werden, um die Geo-Funktionen sowohl als ArcGIS-Extensions wie auch als Web Services nutzbar zu machen. Zu den wesentlichen Vorteilen des RipsFramework zählen die Wiederverwendbarkeit, die zentrale Haltung des Quellcodes, die vereinfachte Wartung sowie die Plattformunabhängigkeit.

In Kapitel 3 wurden die grundlegenden technischen Voraussetzungen für das Rips Web Service Framework vorgestellt. Dazu gehören die Web Service-Technologien (SOAP, WSDL, UDDI), C#.NET und der ESRI ArcGIS Server. Daraufhin erfolgte die prototypische Umsetzung verschiedener Framework-Komponenten. Es wurden Wege aufgezeigt, wie .NET Tools für ArcMap programmiert, und XML Web Services mit dem Application Developer Framework des ArcGIS Server erstellt werden können. Darüber hinaus wurde untersucht, wie diese Dienste unter Verwendung von Proxyklassen in .NET- und MS-Office Clients sowie in HTML/Javascript-Anwendungen eingebunden werden können.

Weitere Schwerpunkte der Ausarbeitung bildeten Untersuchungen hinsichtlich der Verwaltung der RipsWeb-Klassenbibliothek sowie der Kategorisierung und der Veröffentlichung der Webdienste. Es wurde ein Schubladensystem aufgebaut, dass der Klassifizierung der ESRI ArcToolbox entspricht und dadurch offen für zukünftige Erweiterungen ist. Um die Web Services den Anwendern verfügbar zu machen, existieren unterschiedliche Verfahren. UDDI ist weitestgehend standardisiert und wird von vielen großen Unternehmen unterstützt. Im Vergleich zu den proprietären Lösungen von Microsoft bzw. IBM (DISCO und WS-Inspection) ist die Erstellung solcher Verzeichnisse relativ aufwendig und lohnt sich daher nur für eine größere Anzahl an Diensten.

Zudem wurde auf Aspekte der Quellcodeverwaltung und des Teamworking im RipsWeb eingegangen. Nach der Vorstellung verschiedener Versionskontrollsysteme

wurde eine Möglichkeit erarbeitet, wie die Klassenbibliothek zukünftig von Entwicklern der LUBW erweitert werden kann. Das Framework wird in Zusammenarbeit mit der Firma AHK in Freiburg gepflegt und zunächst dort in einem CVS Repository verwaltet. Bei der LUBW wird eine Arbeitskopie zentral abgelegt, die von den Programmierern auf den lokalen Rechner geladen und bearbeitet werden kann. Die abgeschlossene Projektarbeit wird schließlich als gepackte Datei zur Synchronisierung mit dem CVS Server nach Freiburg geschickt.

Abschließend wurden Maßnahmen zur Steigerung der Performance innerhalb der RipsWeb-Architektur diskutiert. Neben Server-seitigen Maßnahmen wie dem Object pooling kann durch den Einsatz von Web Service Caching-Verfahren die Leistungsfähigkeit der Dienste bedeutend verbessert werden.

Mit Abschluss dieser Diplomarbeit wurden verschiedene Grundlagen geschaffen, auf deren Basis das Framework weiterentwickelt werden kann. Die nächsten Schritte umfassen die Erweiterung der RipsWeb- und RipsDesktop-Klassenbibliothek mit Grundfunktionalitäten, die entweder neu programmiert werden müssen, oder aus bereits bestehenden Komponenten anderer LUBW-Projekte übernommen werden können. Hier ist zu beachten, dass die Interaktion zwischen den COM-basierten ArcObjects und .NET in einigen Fällen noch nicht einwandfrei funktioniert. Weitere Gesichtspunkte sind der Ausbau des Exception-Handling, der Web Service-Sicherheit, die Umsetzung der Maßnahmen zur Steigerung der Performance und eine abschließende Vorgehensweise für die Veröffentlichung der Dienste.

9 Anhang

9.1 Glossar

ADF	Application Developer Framework
AHK	Gesellschaft für Angewandte Hydrologie und Kartographie
ALK	Automatisierte Liegenschaftskarte
API	Application Programming Interface
ASP	Active Server Pages
AWGN	Amtliches wasserwirtschaftliches Gewässernetz
BCL	Base Class Library
CLR	Common Language Runtime
COM	Component Object Model
CORBA	Common Object Request Broker Architecture
CVS	Concurrent Versions System
DISCO	Discovery-Dateien
DLL	Dynamic Link Library
DOM	Document Object Model
ESRI	Environmental Systems Research Institute
GAC	Global Assembly Cache
GKZ	Gewässerkennzahl
GUID	Globally Uniquely Identifier
GWD	Gewässerdirektionen
HTML	Hyper Text Markup Language
HTTP	Hyper Text Transfer Protocol
IIS	Internet Information Server
ITZ	Informationstechnisches Zentrum der LUBW
LAN	Local Area Network
LUBW	Landesanstalt für Umwelt, Messungen und Naturschutz Baden-Württemberg
MSDN	Microsoft Developer Network
MSIL	Microsoft Intermediate Language
ODP	Oracle Data Provider for .NET

PGDB	Personal Geodatabase
PHP	Hypertext Preprocessor
RCS	Revision Control System
RIPS	Räumliches Informations- und Planungssystem
RPC	Remote Procedure Call
SDE	Spatial Database Engine
SOA	Service-oriented Architecture
SOAP	Simple Object Access Protocol
SOC	Server Object Container
SOM	Server Object Manager
TCP	Transmission Control Protocol
TLB	Type Library
UDDI	Universal Description, Discovery and Integration
UIS	Umweltinformationssystem Baden-Württemberg
URI	Uniform Ressource Identifier
URL	Uniform Ressource Locator
UVM	Ministerium für Umwelt und Verkehr
VBA	Visual Basic for Applications
WAABIS	Informationssystem Wasser, Abfall, Altlasten, Boden
WASY	Gesellschaft für wasserwirtschaftliche Planung und Systemforschung
WSDL	Web Service Description Language
WSIL	Web Service Inspection Language
XML	Extensible Markup Language

9.2 Literaturverzeichnis

Budszuhn, Frank (2005): CVS. Einführung, Praxis, Referenz, 2. Auflage, Galileo Press GmbH, Bonn.

Doberanz, Walter und Thomas Kowalski (2003): Visual C# .NET. Grundlagen und Profiwissen, Carl Hanser Verlag, München, Wien.

Dostal, Wolfgang; Jeckle, Mario; Melzer, Ingo und Barbara Zengler (2005): Service-orientierte Architekturen mit Web Services. Konzepte-Standards-Praxis, 1. Auflage, Elsevier GmbH (Spektrum Akademischer Verlag), München.

Dustdar, Schahram; Gall, Harald und Michael Hauswirth (2003): Software-Architekturen für Verteilte Systeme. Prinzipien, Bausteine und Standardarchitekturen für moderne Software, Springer Verlag, Berlin, Heidelberg.

ESRI (2004): ArcGIS Server Administrator and Developer Guide. ArcGIS 9.0, ESRI, Redlands (California).

Ferrara, Alex und Matthew MacDonald (2002): Programming .NET Web Services. Building Web Services with ASP.NET and C#, First Edition, O'Reilly & Associates, Sebastopol.

Fraser, Stewart und Steven Livingstone (2002): Beginning C# XML. Essential XML Skills for C# Programmers, Wrox Press, Birmingham.

Freeman, Adam und Allen Jones (2003): Microsoft .NET XML Webdienste. Schritt für Schritt, Microsoft Press Deutschland, Unterschleißheim.

Kersken, Sascha (2005): Handbuch für Fachinformatiker. Der Ausbildungsbegleiter, 2. erweiterte Auflage, Galileo Press GmbH, Bonn.

Kuschke, Michael und Wölfel, Ludger (2002): Web Services kompakt. Spektrum Akademischer Verlag GmbH, Heidelberg, Berlin.

Pomberger, Gustav und Günther Blaschek (1996): Software Engineering. Prototyping und objektorientierte Software-Entwicklung, 2. Auflage, Carl Hanser Verlag, München, Wien.

Pree, Wolfgang (1997): Komponentenbasierte Softwareentwicklung mit Frameworks. 1. Auflage, dpunkt – Verlag für digitale Technologie GmbH, Heidelberg.

Semmler, Monika und Marko Apfel (2002): ESRI und .NET. In: arcAktuell, Ausgabe 3/2002, ESRI Geoinformatik GmbH, Kranzberg.

Szyperski, Clemens (1997): Component Software. Beyond Object-Oriented Programming, Addison-Wesley, Harlow (England), Reading (Massachusetts).

Wenz, Christian (2005): JavaScript. Das umfassende Handbuch, 6. Auflage, Galileo Presss GmbH, Bonn.

Wigard, Susanne (2003): Visual C# .NET. Das bhv Taschenbuch, 1. Auflage, Verlag Moderne Industrie, Buch AG & Co. KG, Landsberg.

9.3 Onlinequellen

APACHE_2006

http://logging.apache.org/log4net/
(aufgerufen am 27.01.2006)

CVS_2006

http://www.nongnu.org/cvs/
(aufgerufen am 02.02.2006)

DEVHELP_2006

http://edndoc.esri.com/arcobjects/9.1/
(aufgerufen am 25.01.2006)

EDN_2006

http://edn.esri.com/
(aufgerufen am 25.01.2006)

ENTWICKLER_2006

http://www.entwickler.de/itr/online_artikel/psecom,id,782,nodeid,31.html
(aufgerufen am 01.02.2006)

ESRI_2006a:

http://www.esri-germany.de/products/arcgis/index.html
(aufgerufen am 25.01.2006)

ESRI_2006b:

http://www.esri.com/software/arcwebservices/index.html
(aufgerufen am 25.01.2006)

GISTERM_2006

http://www2.lfu.baden-wuerttemberg.de/lfu/uis/globus_direkt/globus5/2-ipf/gl02-1.html
http://www.disy.net/disy_gisterm.html
http://www.lubw.bwl.de/local/abt5/itz/rips/gisterm.htm
(aufgerufen am 30.01.2006)

HUNT_2006

http://home.comcast.net/~lancehunt/CSharp_Coding_Standards.pdf
(aufgerufen am 27.01.2006)

JAVACC_2006

http://java.sun.com/docs/codeconv/index.html
(aufgerufen am 27.01.2006)

LUBW_2006a:

http://www.lubw.bwl.de/servlet/is/25639/
(aufgerufen am 25.01.2006)

LUBW_2006b:

http://www.lubw.bwl.de/local/abt5/itz/rips/giswerkzeuge.htm
(aufgerufen am 25.01.2006)

MINDREEF_2006

http://www.mindreef.com
(aufgerufen am 25.01.2006)

MSDN_2006

http://msdn.microsoft.com
(aufgerufen am 25.01.2006)

.NET Design Guidelines
http://msdn.microsoft.com/library/default.asp?url=/library/en-us/cpgenref/html/cpconnetframeworkdesignguidelines.asp
(aufgerufen am 27.01.2006)

Enterprise Library
http://msdn.microsoft.com/library/default.asp?url=/library/en-us/dnpag2/html/entlib.asp
(aufgerufen am 01.02.2006)

WS-Inspection
http://msdn.microsoft.com/library/default.asp?url=/library/en-us/dnglobspec/html/ws-inspection.asp
(aufgerufen am 01.02.2006)

MSVSTUDIO_2005

http://msdn.microsoft.com/vstudio/
(aufgerufen am 25.01.2006)

SCHWICHTENBERG_2006

http://www.it-visions.de/start.aspx
(aufgerufen am 25.01.2006)

STYLECHECK_2006

http://checkstyle.sourceforge.net/
(CheckStyle für Java)
http://jalopy.sourceforge.net/
(Jalopy für Java)
http://www.textrush.com/index.htm
(SourceFormatX)
http://www.synspace.com/DE/Services/syntactics.html
(TICS)
http://www.sqcbw.de/quellcode-strukturierer-formatierer-einruecker-werkzeug-de/
(SQCBW)
(alle Seiten aufgerufen am 27.01.2006)

TIGRIS_2006

http://subversion.tigris.org
(aufgerufen am 25.01.2006)

UDDI_2006

http://uddi.microsoft.com
http://www-3.ibm.com/services/uddi
http://uddi.ariba.com
http://uddi.sap.com
(aufgerufen am 25.01.2006)

WEBLEXIKON_2006

http://www.web-lexikon.de/Bibliothek_(Programmierung).html
(aufgerufen am 31.01.2006)

XMETHODS_2006

http://www.xmethods.net
(aufgerufen am 13.02.2006)

XML&WS_2003

http://www.xmlmagazin.de/itr/online_artikel/psecom,id,376,nodeid,69.html
(aufgerufen am 01.02.2006)

#ZIPLIB_2006

http://www.icsharpcode.net/opensource/sharpziplib/Download.aspx
(aufgerufen am 27.01.2006)

9.4 Tabellenverzeichnis

9.5 Abbildungsverzeichnis

Abbildung 34: **Klassendiagramm DataSource**

Abbildung 35: **Klassendiagramm System**

Abbildung 36: **Basisfunktionen eine GIS**
Bartelme, Norbert (2005): Geoinformatik. Modelle, Strukturen, Funktionen, 4. Auflage, Springer-Verlag, Berlin, Heidelberg, Abbildung 1.10 (geändert).

Abbildung 37: **ArcToolbox und die Klassifizierung der RipsWeb-Web Services**

Abbildung 38: **Benennung der Web Service-Klassen**

Abbildung 39: **Architektur der Microsoft.Uddi-Assemby für .NET**
http://msdn.microsoft.com
(aufgerufen am 25.01.2006)

Abbildung 40: **Hinzufügen einer statischen Discovery-Datei in Visual Studio**

Abbildung 41: **Web Reference auf eine disco-Datei**

Abbildung 42: **Vergleich von UDDI und WS-Inspection**
Dostal, Wolfgang; Jeckle, Mario; Melzer, Ingo und Barbara Zengler (2005): Service-orientierte Architekturen mit Web Services. Konzepte-Standards-Praxis, 1. Auflage, Elsevier GmbH (Spektrum Akademischer Verlag), München, Abbildung 6.1.

Abbildung 43: **WS-Inspection-Datenmodell**
Dostal, Wolfgang; Jeckle, Mario; Melzer, Ingo und Barbara Zengler (2005): Service-orientierte Architekturen mit Web Services. Konzepte-Standards-Praxis, 1. Auflage, Elsevier GmbH (Spektrum Akademischer Verlag), München, Abbildung 6.2.

Abbildung 44: **Verzeichnisdienst von xmethods**
http://www.xmethods.com/
(aufgerufen am 19.01.06)

Abbildung 45: **Informationen zum Service**
http://www.xmethods.com/
(aufgerufen am 19.01.06)

Abbildung 46: **SOAPscope der Firma Mindreef**
http://www.xmethods.com/
(aufgerufen am 19.01.06)

Abbildung 47: **Lokale Arbeitskopie von RipsWeb**

Abbildung 48: **Anpassen der Verweispfade**

Abbildung 49: **Beispiel für die Verwaltung lokaler nutzerrelevanter Projekteigenschaften**

Abbildung 50: **Einrichten eines virtuellen Verzeichnisses**