Diplomarbeit

Web Services

Evaluierung der Integration und Interoperabilität

Ausgeführt zum Zweck der Erlangung des akademischen Grades eines

DI (FH) für Telekommunikation und Medien

am Fachhochschul-Diplomstudiengang

Telekommunikation und Medien St. Pölten

unter der Leitung von

DI Grischa Schmiedl

ausgeführt von

Matthias Bayer

0210038007

St. Pölten, am 31. August 2006 Unterschrift:

Ehrenwörtliche Erklärung

Ich versichere, dass

ich diese Diplomarbeit selbständig verfasst, andere als die angegebenen Quellen und Hilfsmittel nicht benutzt und mich auch sonst keiner unerlaubten Hilfe bedient habe.

ich dieses Diplomarbeitsthema bisher weder im Inland noch im Ausland einem/einer BegutachterIn zur Beurteilung oder in irgendeiner Form als Prüfungsarbeit vorgelegt habe.

Diese Arbeit stimmt mit der von dem/der BegutachterIn beurteilten Arbeit überein.

…….. ……...

Ort, Datum Unterschrift

Kurzfassung

Web Services kommen heute in vielen Bereichen zum Einsatz. Man unterscheidet zwischen zwei verschiedenen Web Service Spezifikationen, die sowohl die Struktur der Anfrage als auch der Antwort bestimmen. Bei den beiden Web Service Standards, die heute hauptsächlich in Verwendung sind, handelt es sich um XML-RPC und SOAP.

Man kann Web Services auf den verschiedensten Systemplattformen einerseits betreiben sowie auch konsumieren. Für viele Plattformen gibt es Implementierungen des jeweiligen Web Service Standards.

Diese Arbeit befasst sich mit den Unterschieden der beiden Web Service Spezifikationen und den Implementierungen die es dafür auf Java, Microsoft .NET und PHP Plattformen gibt. Es werden einige freie Implementierungen der jeweiligen Plattformen ausgewählt und anschließend auf einem Testsystem integriert. Die Implementierungen werden in weiterer Folge auf Interoperabilität getestet.

Die Ergebnisse werden einander im Kapitel 6 gegenübergestellt und sollen die Grenzen der Interoperabilität der einzelnen Implementierungen aufzeigen.

Abstract

Web services appear in many areas these days. There are two different web service specifications, which determine the structure of the web service request and response. The names of these standards are XML-RPC and SOAP.

Web services can be both run and consumed on different system platforms. There are many implementations for both specifications on these platforms.

This document is concerned with the difference of the specifications and the implementation used on Java, Microsoft .NET and PHP platforms. Some free implementations of each platform are chosen and afterwards integrated into the test system. Furthermore, each of the implementations will be tested for interoperability to itself and all other implementations.

The results will be presented and compared in chapter 6 and shall determine the limits of interoperability of each implementation.

Inhaltsverzeichnis

Einleitung

Ob man einen Online Shop bedient, eine Nachricht übers Internet schickt, sich über die letzten Neuigkeiten online informiert oder einfach ein paar Spiele gegen andere Benutzer spielt, man kommt nicht darum herum, Daten auszutauschen.

Bei einfachen Anwendungen wie dem Austausch einer Datei zwischen zwei Personen, die persönlich in Kontakt treten können, ist dies noch relativ einfach. Man speichert die Informationen auf einem Datenträger ab und übergibt diesen anschließend der Zielperson. Die Informationen können jedoch so abgespeichert sein, dass die Zielperson sie nicht öffnen bzw. bearbeiten kann. In diesem Fall hat man sich vorher nicht auf ein Format geeinigt, in dem die Informationen abgespeichert und übergeben werden sollen.

Nachdem heute sehr viel über das Internet passiert, die Daten immer schnelllebiger werden und es noch viel mehr unterschiedliche Formate gibt, in denen man Daten abspeichern kann, kommt es immer häufiger zu Kompatibilitätsproblemen. Ob man die Daten physisch übergibt oder via Internet schickt ist dabei nicht mehr relevant. Viel wichtiger ist jedoch, dass dabei das Format stimmt. Man benötigt also eine gegenseitige Abstimmung der Datenstruktur. Sitzen hinter so einem Datentransfer zwei Menschen, wird dies noch irgendwie machbar sein, jedoch kommt es sehr schnell zu Problemen, wenn dieser Datentransfer von einem automatisierten System ausgeführt wird, das nur seine eigenen Datenformate bzw. Strukturen der Daten kennt. Diesem System wird es nur gelingen gleichen Systemen, bzw. speziell konfigurierten Systemen Daten zuzusenden, bei Fremdsystemen wird der Datentransfer nicht stattfinden.

1. Web Services

1.1. Was sind Web Services?

Es gibt viele verschiedene Plattformen in der Welt der IT. In einer Zeit, in der man versucht, alles miteinander zu vernetzen, führen diese Unterschiedlichkeiten der teilnehmenden Systeme unweigerlich zu Kommunikationsproblemen. Das Internet bietet hier zwar ein sehr weit verbreitetes Protokoll (HTTP – HyperText Transfer Protocol), welches das Fundament bereitstellt, über das man Verbindungen untereinander herstellen kann, jedoch ist es allein durch diese Bereitstellung noch nicht getan. Geeignete Schnittstellen (engl. Interface) müssen definiert werden, die es unterschiedlichsten Plattformen ermöglichen, Informationen auszutauschen.

Aus diesem Grund befasst sich diese Arbeit mit einer der Möglichkeiten plattformübergreifend Daten auszutauschen, nämlich Web Services.

1.2. Wie funktionieren Web Services?

Web Services kann man als vom Computer gelesene Webseiten verstehen, bei denen Daten in XML (Extensible Markup Language, siehe 1.2.1) Format von einer Applikation zu einer anderen Applikation über ein Protokoll (zum Beispiel über TCP – Transmission Control Protocol) gesendet werden. Die heute üblichen und in dieser Arbeit behandelten XML Formate, in dem diese Daten versendet werden, sind entweder nach XML-RPC oder nach SOAP (W3C Standard) spezifiziert (siehe 1.4).

Die allgemeine Funktionsweise von Web Services ist wie folgt.

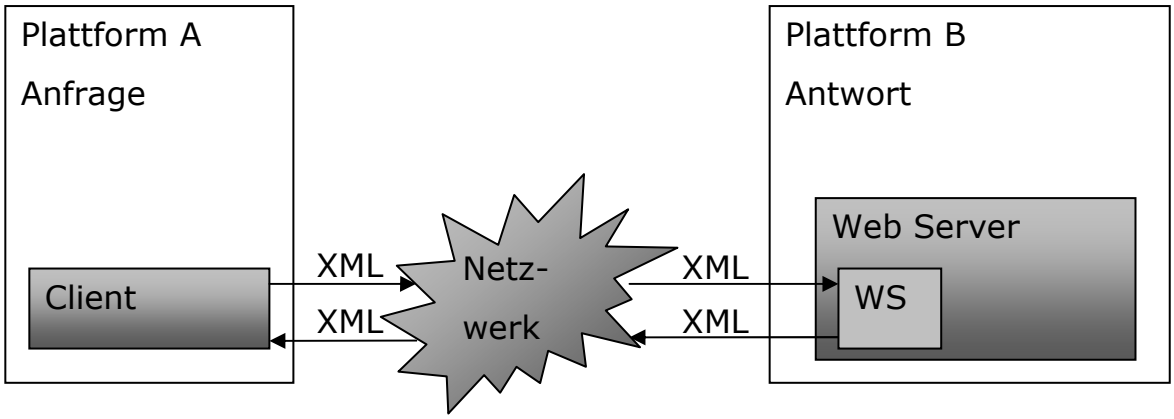

Abbildung 01 – Web Service Kommunikation Allgemein

Ein üblicherweise entfernter Rechner möchte Informationen über ein Netzwerk, wie zum Beispiel dem Internet, anbieten. Zunächst muss entschieden werden, welche Spezifikation benutzt werden soll. Ist diese Wahl getroffen, so richtet man einen Web Server, mit einer für die Programmiersprache entwickelten Implementierung der jeweiligen Spezifikation, ein. Dieser Web Server bietet das Web Service über eine Internetadresse, einer so genannten URI (Uniform Resource Identifier), an. Eine andere Applikation kann nun dieses Web Service aufrufen und Daten austauschen. Der Aufruf muss der Spezifikation entsprechen, die auf dem ersten Web Server verwendet wird. Da die Daten in beide Richtungen in XML Format umgewandelt werden, ist es irrelevant welche Sprache der Aufrufer (Client) und welche der Anbieter (Server) verwendet, solange die jeweilige Sprache eine Implementierung der verwendeten Spezifikation unterstützt.

1.2.1. XML

XML ist ein vom W3C (World Wide Web Consortium, siehe 1.4.2.2) definierter Standard zur Strukturierung von Daten und steht für „eXten-

sible Markup Language". Die Daten liegen dabei im Klartext vor, was ein leichtes Lesen und Editieren ermöglicht, ohne dass ein spezielles Programm dafür eingesetzt werden muss. Das Hauptanwendungsgebiet von XML liegt im „Austausch von Daten und Dokumenten zwischen Plattformen, Applikationen und Organisationen". Ein XML Dokument kann prinzipiell beliebig gefüllt werden, allerdings muss man sich an gewisse Strukturen halten.
(vgl. Weyer, 2002, S162)

Ein XML Dokument muss am Beginn stets als solches deklariert werden. Die Versionsnummer ist dabei Pflicht, während die Art der Zeichencodierung optional ist.

```
<?xml version=„1.0“ encoding=„UTF-.8“?>
```

Die eigentlichen Daten bestehen aus Elementen, welche die grundlegende Struktur eines Dokuments definieren. Die Elemente müssen immer einen Anfangstag und einen Endtag aufweisen und dürfen sich hierarchisch nicht überkreuzen.

Richtig:

```
<content>
    <text>
        Textbeispiel
    </text>
</content>
```

Falsch:

```
<content>
    <text>
        Textbeispiel
</content>
    </text>
```

Tags können zusätzlich auch Attribute haben, man kann sie als „Metadaten eines Elements" bezeichnen. Attribute geben also weiterführende

Informationen über das Element, dem sie zugeteilt werden. Wichtig ist, die Werte der Attribute unter Anführungszeichen zu setzen.
(vgl. Weyer, 2002, S163)

```
<Element Attribut=„Wert“>Elementtext</Element>
```

Ausnahmen bieten leere Tags, die mit einem „/“ vor der geschlossenen spitzen Klammer nur ein Tag aufweisen:

```
<text/>
```

XML findet heute in vielen Bereichen seine Anwendung, so auch bei Web Services.

1.3. Warum gibt es Web Services?

Es ist unmöglich, dass man sich auf eine genormte Plattform einigt, da in der Wirtschaft einfach zu viele verschiedene Systeme im Umlauf sind. Auch die Spanne der Medien, die miteinander Daten austauschen sollen, wird immer größer. Mit der Kommunikation über Web Services wird versucht, sich auf einen gemeinsamen Standard zu einigen. Web Services haben gegenüber ihren Konkurrenten DCOM, RMI und CORBA den Vorteil, dass die Nachrichten ausschließlich in XML geschrieben sind und somit vollkommen plattform- und programmiersprachenunabhängig sind. Außerdem erfolgt die Datenübertragung über HTTP Port 80 und ist daher für Firewalls meistens kein Problem.
(vgl. Cerami, 2002, S43)

Web Services werden in vielen Bereichen eingesetzt, sei es nun im Bereich der Web Sites oder im multimedialen Bereich.

Die bekannte Suchmaschine „Google" bietet zum Beispiel ein Web Service an, mit dem User die Möglichkeit haben, die Suchfunktion, die Google auf ihrer Webseite selbst anbietet, auch auf der eigenen Seite einzubauen und die Ergebnisse grafisch an das eigene Layout anzupassen.

Web Services finden ihren Einsatz genauso im IPTV (Internet Protocol Television) Bereich. Im Rahmen eines Fachhochschulprojekts wurde für ein Hotel in Krems eine Plattform geschaffen, die es erlaubt über den Fernseher in den Zimmern Funktionen, wie das Adaptieren der Raumeinstellungen, das Reservieren im Spa Bereich und das Abrufen von Touristeninformationen, bereitzustellen. Auch externe Angebote, wie ein Ticketing System oder ein Touristenführer, werden Kunden bereitgestellt. Bei diesem Projekt sind beide der heute gängigen Spezifikationen zum Einsatz gekommen, nämlich SOAP und XML-RPC. So konnten extern gelagerte Programme bzw. Programmteile auf relativ unkompliziertem Weg angesprochen werden.

Eine wichtige Anwendung finden Web Service heute auch im Middleware Bereich. Unter Middleware versteht man Software, die eine Schnittstelle zwischen zwei oder mehreren Anwendungen anbietet. Web Services haben dabei die Aufgabe, die Kommunikation zwischen der Middleware und der der jeweiligen Anwendung zu übernehmen. Notwendig ist dies unter anderem, wenn es in einem Betrieb mehrere unterschiedliche Softwaresysteme auf Basis unterschiedlicher Programmiersprachen gibt, die es zu vernetzen gilt.

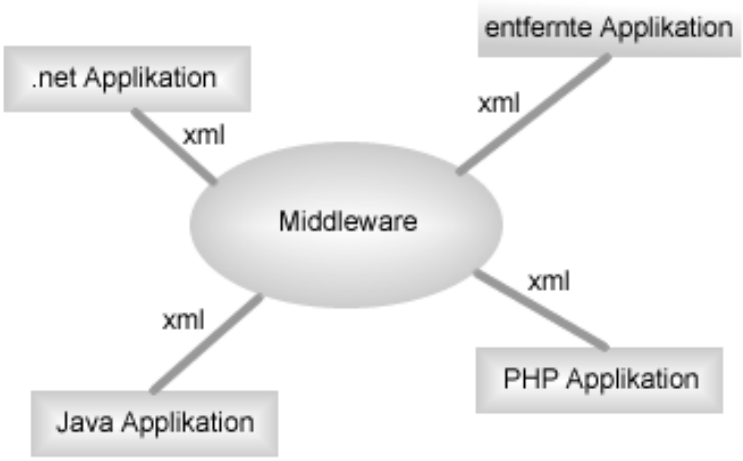

Abbildung 02 – Middleware

Die hier dargestellten Beispiele dienen zur Orientierung, warum Web Services heute so oft zum Einsatz kommen. Gründe dafür sind neben der schon erwähnten Datenübertragung über HTTP Port 80 auch die Tatsachen, dass offene Standards wie SOAP oder XML-RPC keine Lizenzkosten verursachen und gleichzeitig der Umgang mit XML keinen allzu großen Lernaufwand voraussetzt.

1.4. Spezifikationen

Die zwei am häufigsten eingesetzten Spezifikationen im Web Service Bereich sind XML-RPC und SOAP. Spezifikationen für den Nachrichtenversand sind deshalb notwendig, um auch wirklich plattformübergreifend arbeiten zu können. Es müssen gemeinsame Schnittstellen geschaffen werden, die ein Austauschen von Nachrichten ermöglichen. Es gibt Organisationen, die sich rund um dieses Thema gebildet haben, wie das W3C (World Wide Web Consortium, siehe Kapitel 1.4.2.2) welches für den SOAP Standard verantwortlich ist und die WS-I (Web Service Interoperability Organisation), die sich um die Interoperabilität zwischen Web Services bemüht.

1.4.1. XML-RPC

XML-RPC steht für „Extensible Markup Language – Remote Procedure Calling" und ist eine Schnittstellendefinition für Web Services.

„XML-RPC ist eine Spezifikation, die es unterschiedlichen Umgebungen auf verschiedenen Betriebssystemen erlaubt, entfernte Methoden über das Internet aufzurufen.

Die Aufrufe der entfernten Prozeduren verwenden HTTP für den Transport und XML für das Enkodieren. XML-RPC ist darauf ausgelegt, so einfach wie möglich zu sein, während komplexe Datenstrukturen übertragen, bearbeitet und zurückgeschickt werden können."
(http://www.xmlrpc.com/, 07.05.2006)

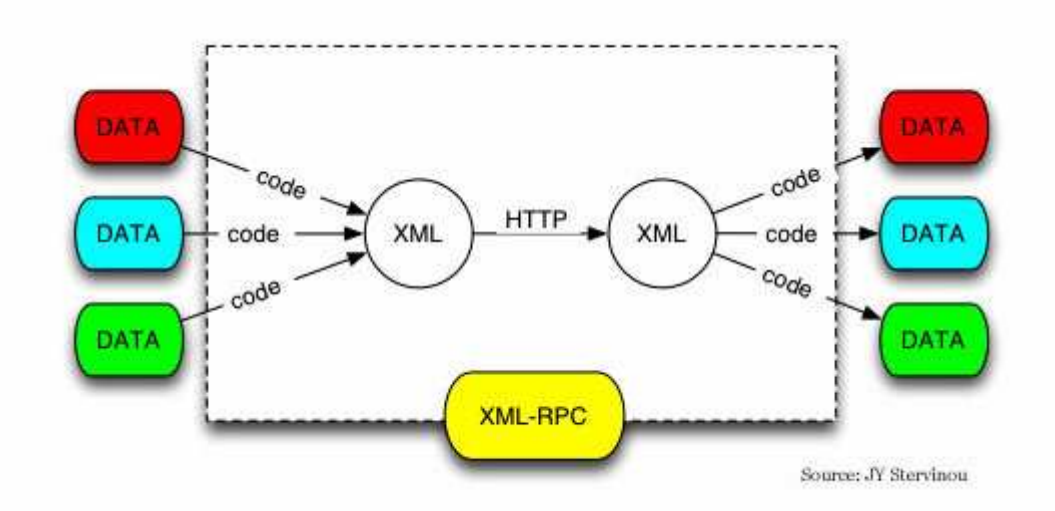

Abbildung 03 – XML-RPC Kommunikation

1.4.1.1. Die Geschichte von XML-RPC

Dave Winer kam 1998 auf die Idee, eine Verbindung zwischen XML und HTTP zu schaffen. Diese Verbindung sollte dafür genutzt werden, XML Nachrichten über das Internet zu senden. Bereits zu diesem Zeitpunkt hatte Winer Kontakt zum COM-Team bei Microsoft, das in Folge sehr

stark an der SOAP Entwicklung beteiligt war (siehe 1.4.2). Am 4. April 1998 wurde die erste Version von XML-RPC veröffentlicht. Diese Version wurde in das Produkt „Frontier 5.1" der von Dave Winer gegründeten Firma „UserLand" implementiert. Nach der Veröffentlichung bildeten sich bereits die ersten Gemeinschaften um XML-RPC, das sehr schnell auf unterschiedlichsten Systemen seine Anwendung fand. Der Standard wurde 1999 um einige Anregungen aus der Community erweitert und Fehler wurden behoben. Nach dem Jahr 1999 verhielt es sich ruhig um XML-RPC Neuerungen, erst im Jahr 2003 wurde eine weitere, letzte Änderung vorgenommen. Es wurde die Beschränkung eines Strings auf ASCII (American Standard Code for Information Interchange) Enkodierung aufgehoben.
(vgl. Hauser, Löwer, 2003, S30-31)

1.4.1.2. Aufbau eines XML-RPC Requests

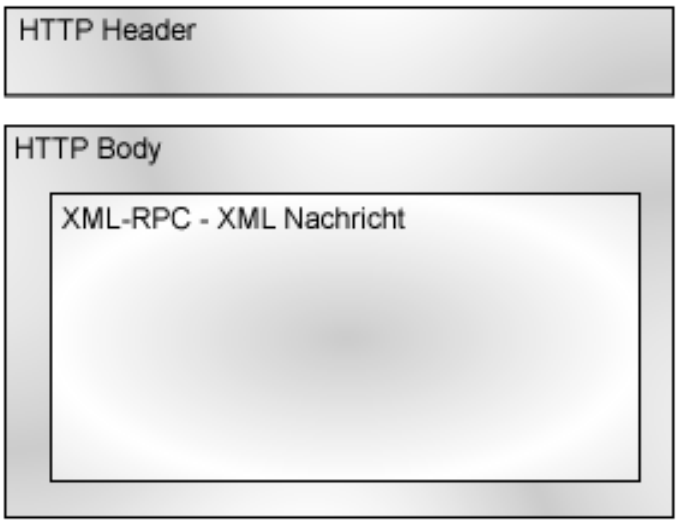

Abbildung 04 – Aufbau einer XML-RPC Nachricht

Eine XML-RPC Nachricht ist ein HTTP Request und besteht daher aus einem HTTP Header (beinhaltet Informationen über die Nachricht) und einem HTTP Body (beinhaltet die eigentliche Nachricht).

Der HTTP Header kann zum Beispiel so aussehen:

```
01  POST /RPC2 HTTP/1.1
02  User-Agent: FH-St. Pölten/1.0 (WinNT)
03  Host: weblab1.fh-stpoelten.ac.at
04  Content-Type: text/xml
05  Content-length: 196
```

Code 01 – HTTP Header

Die Spezifikation (http://www.xmlrpc.com/spec) gibt an, dass ein User-Agent und ein Host angegeben werden muss, der Content-Type muss „text/xml" sein und auch die Content-length muss die korrekte Länge des Inhalts aufweisen. Die Erstellung dieses HTTP Headers übernimmt die jeweilige Implementierung beim Absenden der Daten automatisch.
(vgl. St. Laurent, Johnston, Dumbill, 2001, S25-28)

Die eigentliche Information liegt im Body der Nachricht. Dieser ist in XML Format aufgebaut und kann unterschiedlich viele <param> Tags enthalten.

```
01  <?xml version="1.0"?>
02  <methodCall>
03     <methodName>students.getStudentName</methodName>
04     <params>
05        <param>
06           <value><string>tm021007</string></value>
07        </param>
08     </params>
09  </methodCall>
```

Code 02 – HTTP Body

Die Anfrage ruft eine Methode namens „students.getStudentName" auf dem Zielserver auf. Die Parameter (<param>) Tags haben jeweils ein Element (im Beispiel ist dieses Element <value>), das mit übergeben wird. <value> kann aus unterschiedlichen Werttypen bestehen, nämlich Integer, Boolean, Double, DateTime, Base64 oder String, die wie-

derum in Tags angegeben werden. Wird kein Typ angegeben, so wird der Typ String angenommen.

Anstatt dem Element <value> können auch die Elemente <array> und <struct> übertragen werden. In letzterem kann man auch Name – Wert Paare definieren.

Weiterführende Informationen sind der offiziellen Website zu entnehmen: http://www.xmlrpc.com/.

1.4.2. SOAP

SOAP (ehemals „Simple Object Access Protocol") ist eine weitere in XML geschriebene Spezifikation, die über Netzwerkprotokolle wie dem HTTP, übertragen wird. Die SOAP Spezifikation kann als Rahmenwerk für die Übertragung der XML-Nachrichten verstanden werden.
(vgl. Weyer, 2002, S167)

1.4.2.1. Die Geschichte von SOAP

„Über die Ursprünge von Web-Services kursieren unterschiedliche Anekdoten. Don Box, einer der Miterfinder von SOAP und heute Mitarbeiter von Microsoft, erklärt ihre Entstehung damit, dass er und einige andere Fachbuchautoren herausfinden wollten, wer von ihnen den höchsten Rang in der Verkaufsliste von Amazon einnimmt. Zu diesem Zweck wollten sie aber nicht ständig die Web-Site des Online-Händlers besuchen, sondern die betreffenden Informationen maschinell auslesen. Allerdings bot Amazon seine Kataloge damals nur als HTML-Seiten an, so dass derartige Scripts die gewünschten Informationen aus Bergen von Layout-Angaben extrahieren mussten. Deshalb entstand der

Wunsch, Inhalte in Form strukturierter Daten gezielt aus dem Web abrufen zu können."
(http://www.computerwoche.de/541408, 25.09.2003)

So wurden im Jahr 1999 die Versionen 0.9 und 1.0 veröffentlicht, wobei die Reaktionen auf die SOAP Spezifikation Anfangs sehr zurückhaltend waren. Erst als sich 2000 IBM an der Entwicklung von SOAP beteiligte und die Version SOAP 1.1 beim WWW Konsortium (W3C, siehe 1.4.2.2) einreichte, steigerte sich das allgemeine Interesse. Um SOAP weiterzuentwickeln wurde eine Arbeitsgruppe eingerichtet, welche als Ergebnis die Version 1.2 im Juni 2003 hervorbrachte. Seit dieser Version steht SOAP nicht länger für „Simple Object Access Protocol", was zweierlei Gründe hat. Einerseits hielt man den Namen nicht mehr passend für das umfangreiche Protokoll und andererseits kann man Abkürzungen in den USA nicht schützen lassen, SOAP als Name hingegen schon.
(vgl, http://de.wikipedia.org/wiki/SOAP, 16.05.2006)

1.4.2.2. W3C

„Das World Wide Web Consortium (W3C) ist ein internationales Konsortium, in dem Mitgliedsorganisationen, ein Stab von Vollzeitmitarbeitern und die Öffentlichkeit gemeinsam daran arbeiten, Web-Standards zu entwickeln. Das W3C verfolgt seine Ziele hauptsächlich durch die Erstellung von Web-Standards und Richtlinien, die ein langfristiges Wachstum des Web sicherstellen sollen. Über 400 Organisationen sind Mitglieder des Konsortiums. W3C wird gemeinsam vom MIT Computer Science and Artificial Intelligence Laboratory (MIT CSAIL) in den USA, der European Research Consortium for Informatics and Mathematics (ERCIM) mit Sitz in Frankreich und der Keio University in Japan geführt und hat darüber hinaus ein weltweites Netz von W3C-Büros."
(Tijwinski, 2006, 07.05.2006)

1.4.2.3. Aufbau eines SOAP Requests

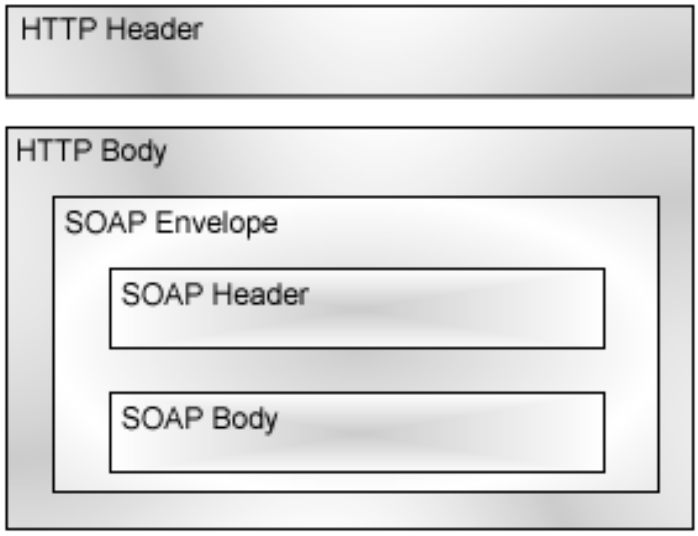

Abbildung 05 – Aufbau einer SOAP Nachricht

Die Nachricht wird in einem so genannten SOAP Envelope im HTTP Body transportiert. Unter SOAP Envelope kann man sich einen Behälter (bzw. Umschlag) vorstellen, in dem die erstellte Nachricht samt Zusatzinformationen zu finden ist. Der SOAP Envelope besteht aus zwei Teilen, dem SOAP Header und dem SOAP Body. Der SOAP Header ist optional, während der SOAP Body auf jeden Fall in einem SOAP Envelope zu finden sein muss. Im Body finden sich die eigentlichen Informationen, die übertragen werden. Im Header sind zusätzliche Daten zu finden wie zum Beispiel Benutzerdaten (Benutzername und Passwort) für eine Anfrage die mittels SOAP gestellt wird.
(vgl. Weyer, 2002, S167ff)

```
01  <SOAP-ENV:Header>
02     <sec:credentials
    Xmlns:sec=„http://schemas.xmlsoap.org/ws/2002/01/sec“
03        mustUnderstand=„1“>
04        <sec:username>CWeyer</sec:username>
05        <sec:userPassword>w3efr56z</sec:userPassword>
06     </sec:credentials>
07  </SOAP-ENV:Header>
```

Code 03 – SOAP Header

(Weyer, 2002, S168)

Man sieht deutlich, dass SOAP mit Namensräumen (Namespaces) arbeitet. Der Parameter „mustUnderstand“ ist dafür zuständig, dass der SOAP Header von der aufrufenden Applikation auf jeden Fall abgearbeitet wird.

```
01  <SOAP-ENV:Body>
02     <SOAP:GetUserID
03        xmlns:SOAP=„http://eyesoft.de/webservices/“>
04        <userName>CWeyer</userName>
05     </SOAP:GetUserID>
06  </SOAP-ENV:Body>
```

Code 04 – SOAP Body

Die Daten werden zur Übertragung serialisiert, das bedeutet in XML Format umgewandelt und so übertragen. Die Daten werden beim Empfänger wieder aus dem XML ausgelesen und verarbeitet.

1.4.2.4. WSDL

Wenn eine Applikation ein Web Service ansprechen möchte, dann weiß diese im Vorhinein nicht welche Funktionen das Web Service bereitstellt, welche Parameter es benötigt, es weiß nicht einmal wofür das Web Service überhaupt zuständig ist. Deswegen gibt es das WSDL (Web Service Description Language) File. Dabei handelt es sich um ein XML Format, das, wie der Name schon sagt, eine Beschreibung für das

Web Service darstellt. Es werden unter anderem Parameter definiert, welche die Schnittstellen des Web Services entgegen nehmen. Weiters ist dort definiert, was vom Web Service zurückgeliefert wird. Mit Hilfe des WSDL Files ist es möglich, Web Services automatisiert zu nutzen, indem man die WSDL Datei einliest und daraus automatisch einen Proxy generieren lässt. Über diese Proxy Klasse kann man das Web Service wie ein lokales Objekt benutzen, die Datenübertragung im SOAP Format übernimmt die Proxy Klasse mit Hilfe der Spezifikationsbibliotheken der Implementierung. WSDL gibt es nur für SOAP und nicht für XML-RPC, daher ist man bei letzterem auf eine gute Dokumentation zu den Web Services angewiesen. Wie das WSDL erstellt wird hängt von der verwendeten Implementierung ab und wird in den jeweiligen Integrationskapitel im Detail behandelt. Kurz zusammengefasst erzeugen einige Implementierungen das WSDL File automatisch, während man die WSDL Datei bei anderen manuell erstellen muss.
(vgl. Englander, 2002, S170-171)

1.4.2.5. UDDI

Das Universal Beschreibungs-, Entdeckungs- und Integrationsprotokoll UDDI (Universal Description, Discovery Integration) ist ein optionaler Bestandteil im Web Service Prozess. Es handelt sich dabei um eine „interoperable Plattform", die Web Services im Internet schnell und einfach findet und benutzbar macht.
(vgl. http://www.uddi.org/about.html, 16.05.2006)

Man kann sich das ganze wie ein Telefonbuch vorstellen, bei dem sich Anbieter eines Dienstes eintragen können. Diese Dienste können sowohl von Firmen als auch von Kunden genutzt werden und dienen dazu, den Bekanntheitsgrad des angebotenen Web Services zu steigern. Für das Auffinden von Web Services gibt es eigene Suchmaschinen.

Dass UDDI immer mehr in Vergessenheit gerät hat zweierlei Gründe. Zum einen, dass viele Firmen nicht verstehen wofür UDDI genau zuständig ist und zum anderen, dass man sich in der Vergangenheit beim Erstellen eines Eintrags als jemand anderer, wie zum Beispiel als „Microsoft", ausgeben konnte, was der Reputation von UDDI natürlich merklich geschadet hat.

(vgl. Tidwell, Snell, Kulchenko, Programming Web Services with SOAP, 2001, S155-156)

1.5. Hintergründe zu Integrationstechnologien

1.5.1. Historisches: Die Welt vor Web Services

Web Services bieten eine relativ junge Variante der Kommunikation zwischen Middleware und den zu verbindenden Plattformen. Die Möglichkeit des Vernetzens gibt es schon sehr lange. Im Folgenden werden die bekanntesten Technologien, nämlich SunRPC, CORBA, DCOM und RMI, beschrieben.

1.5.1.1. Remote Procedure Calls (SunRPC)

RPC war ursprünglich eine Entwicklung von Sun (http://www.sun.com/) und für den Einsatz von NFS (Network File System) gedacht. Es handelt sich bei RPC um eine „Technik zur Netzwerkkommunikation". Genauer gesagt können Funktionen auf einem entfernten Rechner über ein Netzwerk aufgerufen werden. Die Art der Kommunikation ist „synchron", das bedeutet, dass der Client nach erster Kontaktierung auf die Rückmeldung des Servers wartet, bevor er mit dem Programmcode fortsetzt.

(vgl. http://de.wikipedia.org/wiki/Remote_Procedure_Call, 12.05.2006)

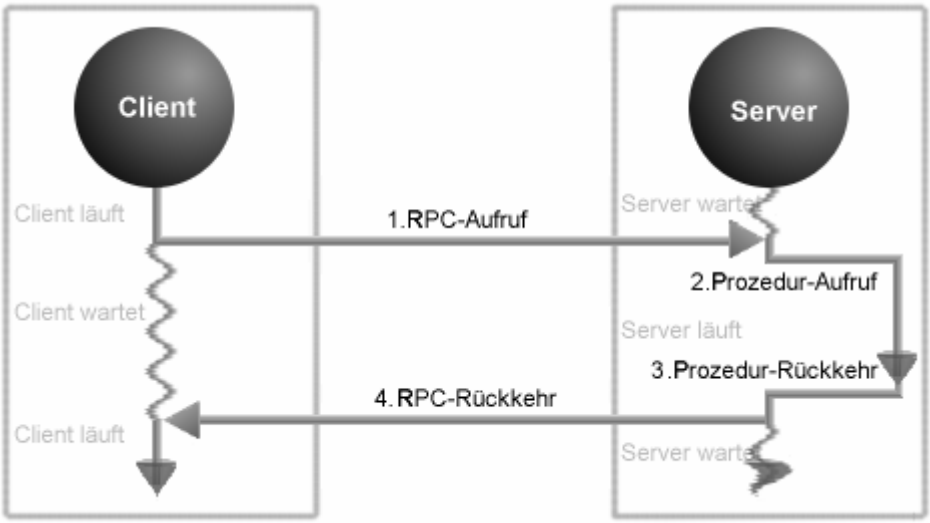

Abbildung 06 – Ablauf eines RPC-Aufrufs
(http://www.linuxfibel.de/rpc.htm, 12.05.2006)

Zwar bauen Web Services auf dem Prinzip von RPC auf, jedoch gibt es deutliche Unterschiede, wie zum Beispiel den durch XML abgelösten Kommunikationsstandard XDR (eXternal Data Representation). Diverse Implementierungen für SunRPC sind außerdem anfällig für Integer Overflows in ihren Bibliotheken, daher empfiehlt es sich RPC nicht über das Internet zu verwenden.
(vgl. http://www.linux-magazin.de/Artikel/ausgabe/2003/06/024_news_insec/insec_news.html, 27.07.2006)

1.5.1.2. CORBA – Common Object Request Broker Architecture

Die Common Object Request Broker Architecture, kurz CORBA, definiert plattformübergreifende Protokolle und Dienste. Sie wurde von der Object Management Group (OMG) entwickelt und dient dazu das Erstellen verteilter Anwendungen in heterogenen Umgebungen zu vereinfachen.
(http://de.wikipedia.org/wiki/CORBA, 09.05.2006)

CORBA wurde und wird auch heute noch dort eingesetzt wo unterschiedliche Applikationen miteinander arbeiten müssen. Ende der

1990er Jahre galt CORBA als die „ultimative Referenz für Middleware schlechthin". Heute hat es aber deutlich an Bedeutung verloren, wenn gleich es nach wie vor in manchen Firmen eingesetzt wird. Einer der Gründe warum sich CORBA nicht halten konnte war die recht komplizierte Einbindung in die hauseigenen Systeme, aufgrund eines sehr „schwergewichtigen Mechanismus, der für Entwickler nicht sehr benutzerfreundlich ist".
(vgl. Kapp, 2003, 11.05.2006)

1.5.1.3. DCOM – Distributed Component Object Model

DCOM steht für Distributed Component Object Model und ist eine Erweiterung von COM (Component Object Model), die COM Objekten erlaubt über ein Netzwerk zu kommunizieren. Dabei wird der RPC Mechanismus benutzt, um Informationen zwischen COM Objekten über das DCOM Protokoll auszutauschen. COM Objekte sind eine Entwicklung von Microsoft und das erste Mal in Windows NT 4 implementiert worden. Es gibt auch für andere Plattformen entsprechende Adaptierungen um DCOM zu nutzen, wie zum Beispiel JCOM für Java. Heute ist DCOM auch in der „Microsoft-Welt" von SOAP größtenteils abgelöst worden.
(vgl. http://www.webopedia.com/TERM/D/DCOM.html, 03.06.2006)

1.5.1.4. RMI – Remote Method Invocation

Remote Method Invocation (RMI) bezeichnet den Aufruf einer Methode eines entfernten Java-Objekts und ist sozusagen das Pendant der Java Welt zu DCOM in der Microsoft Welt.

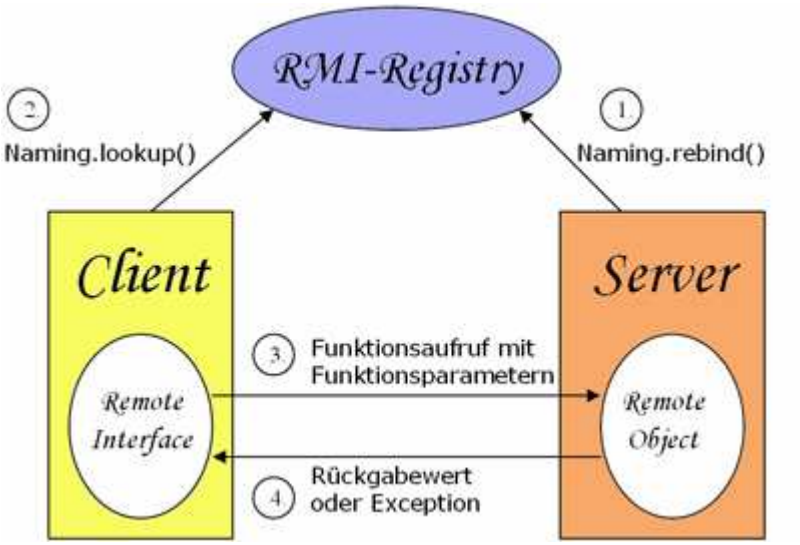

Abbildung 07 – Remote Method Invocation

1. „Der Server registriert ein Remote Object bei der RMI-Registry unter einem eindeutigen Namen."
2. „Der Client schaut bei der RMI-Registry unter diesem Namen nach und bekommt eine Objektreferenz, die seinem Remote Interface entsprechen muss."
3. „Der Client ruft eine Methode aus der Objektreferenz auf (dass diese Methode existiert, wird durch das Remote Interface garantiert)."
4. „Der Server gibt dem Client die Rückgabewerte dieses Aufrufes, oder der Client bekommt eine Fehlermeldung (z. B. bei einem Verbindungsabbruch)."

(http://de.wikipedia.org/wiki/Remote_Method_Invocation, 03.06.2006)

1.5.2. EAI – Enterprise Application Integration

Enterprise Application Integration ist weit mehr als nur eine Plattform zum Datenaustausch. Bei EAI handelt es sich eher um einen Themenbereich als um ein Produkt, also ein Begriff für die Gesamtheit aller Ansätze, Technologien, Protokolle, etc. Dabei geht es auch sehr stark in den konzeptionellen Bereich, wo EAI für das Planen neuer Funktionen oder Anwendungen benutzt wird. EAI kann mit allen vorher beschriebe-

nen Ansätzen realisiert werden, seien es nun Web Services, RMI, DCOM, etc.

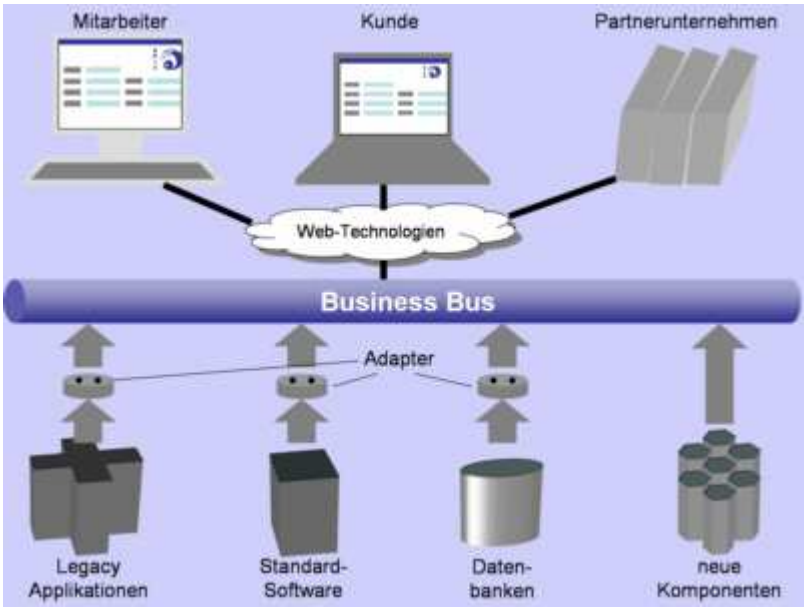

Abbildung 08 – Enterprise Application Integration

(vgl. http://de.wikipedia.org/wiki/Enterprise_Application_Integration, 03.06.2006)

2. Problembenennung

2.1. Welche Plattform ist die Richtige?

Es gibt mittlerweile eine Vielzahl von Technologien mit denen man plattformübergreifend arbeiten kann. Von Middleware über EAI bis zu Web Services oder auch Kombinationen daraus stehen einem mehrere Möglichkeiten zur Verfügung. Die Vielfalt ist damit aber noch lange nicht zu Ende, wie es Web Services beweisen, auf die sich dieses Werk spezialisiert. Es bleibt die Frage, auf welche Art man sein Web Service erstellen will. Web Services können von vielen Programmiersprachen implementiert werden. Es gibt auch eigene Entwicklungsumgebungen, die automatisierte Funktionen bereitstellen um das Erstellen von solchen Diensten zu erleichtern. Diese Arbeit wird sich auf drei der gängigsten Programmiersprachen in diesem Bereich fokussieren, nämlich Java, PHP und C#. Es wird hierbei untersucht, wie aufwändig es aktuell ist, ein lauffähiges Web Service auf drei verschiedenen Plattformen zu erstellen und welche Unterstützung von den getesteten Implementierungen dafür zur Verfügung steht. Anschließend soll überprüft werden, ob Modifikationen erforderlich sind um die unterschiedlich hergestellten Web Services mit allen verwendeten Plattformen sprechen zu lassen.

2.2. Welche Spezifikation ist die Richtige?

Es soll untersucht werden, welche Vor- und Nachteile die hauptsächlich eingesetzten Standards SOAP und XML-RPC haben. Es ist nicht immer im Vorhinein klar welcher Standard für welche Zwecke geeigneter ist. Obwohl SOAP eine Weiterentwicklung von XML-RPC ist, hat letzteres immer noch eine große Bedeutung in der Wirtschaft.

2.2.1. SOAP ist nicht gleich SOAP

Im WSDL File ist definiert wie eine SOAP Nachricht verpackt und verarbeitet wird. Ist kein WSDL File vorhanden oder lässt die Implementierung nur eine Art der Verpackung zu, so entscheidet die jeweilige Implementierung welche Verpackungsart zu wählen ist, was in folge zu Inkompatibilitäten führen kann.

In der Praxis gibt es zwei Möglichkeiten wie eine Nachricht verpackt wird, nämlich „RPC/Encoded" und „Document/Literal".

Der „RPC/Encoded" Ansatz geht davon aus, dass man wie bei den herkömmlichen RPC (siehe 1.5.1) Technologien wie beispielsweise RMI, CORBA oder DCOM entfernte Prozeduren aufruft. Man sendet den Methodenname und die dazugehörigen Parameter an den Zielrechner und erhält als Rückgabewerte die Ausgabeparameter zurück. Die Struktur der Sendung wird durch den „Encoded" Teil des Ansatzes definiert und ist in der SOAP 1.1 in Sektion 5 genormt.

(vgl. http://msdn.microsoft.com/webservices/webservices/understanding/webservicebasics/default.aspx?pull=/library/en-us/dnwebsrv/html/rpc_literal.asp, 27.05.2006)

Bei der „Document/Literal" Kodierung wird ein Dokument an den Zielrechner übergeben, das von diesem ausgewertet wird. Für die Struktur dieser SOAP Nachricht gibt es keine fixe Definition, jedoch wird im Normalfall ein XML-Schema verwendet. Aufgrund des verwendeten XML-Schemas kann man das übertragene Dokument auf korrekte Syntax validieren. Der Nachteil ist natürlich, dass durch das Komplexitätspotential des Dokuments für den Hersteller ein höherer Aufwand bei der Implementierung neuer Integrationsprodukte entsteht.

(vgl. http://www.innoq.com/blog/st/2003/03/web_servicesstile_dokument_vs_rpcorientiert.html, 26.07.2006)

Der Nachrichtenstil (RPC bzw. Dokument) wird im <soap:binding /> Tag der WSDL Datei definiert, in der auch das verwendete Transport Protokoll zu finden ist. Die Art der Enkodierung (Encoded bzw. Literal) der einzelnen Parameter wird im <soap:body /> Tag spezifiziert.
(vgl. Tidwell, Snell, Kulchenko, Programming Web Services with SOAP, 2001, 88)

Welcher Nachrichtenstil verwendet wird, hängt von der Implementierung ab, die man benutzt. Vor allem ältere Implementierungen unterstützen nur RPC/Encoded, einige neuere Implementierungen, wie zum Beispiel Axis, unterstützen beide Varianten und andere, wie zum Beispiel Microsoft's .NET Implementierung, gehen in die Richtung Document/Literal, da dieser Nachrichtenstil WS-I konform ist.

3. Forschungsleitende Fragestellung

Die Forschungsleitende Fragestellung ist in drei Fragen gegliedert. Die erste Frage befasst sich mit der Integration der Implementierungen. Die zweite Frage ist die Hauptfragestellung und fragt nach der Interoperabilität der Implementierungen. Die dritte Frage behandelt die Vor- und Nachteile der Web Service Spezifikationen XML-RPC und SOAP.

3.1. Forschungsfrage nach der Integration

Wie unterscheiden sich verschiedene SOAP/XML-RPC Implementierungen unterschiedlicher Programmiersprachwelten von einander im Hinblick auf die Erstellung von Web Services?

Untersucht wird hier, wie man auf unterschiedlichen Plattformen Web Services erstellen und anwenden kann. Die Ergebnisse werden anschließend interpretiert und so weit wie möglich gegenüber gestellt.

3.2. Forschungsfrage nach der Interoperabilität

Welche Diskrepanzen treten auf, wenn man mit unterschiedlichen Programmiersprachen eine einheitliche, interoperable Funktion durch Web Services erreichen will?

Hierbei sollen die in der Problemstellung erörterten Abnormitäten untersucht werden, die bei der Plattformunabhängigkeit von Web Services auftreten können.

3.3. Forschungsfrage nach der Spezifikation

Warum wird heutzutage immer noch mit XML-RPC gearbeitet, obwohl SOAP eine daraus resultierende Weiterentwicklung darstellt?

Es soll herausgefunden werden, warum XML-RPC noch immer so eine große Anhängerschaft hat, obwohl SOAP auf dem XML-RPC Standard basiert und im Gegensatz zu XML-RPC noch immer weiterentwickelt wird.

3.4. Hypothesenbildung

Aufgrund der Forschungsmethode (Inhaltsanalyse) werden Hypothesen erst am Ende der Arbeit aufgestellt.

4. Forschungsstrategie

4.1. Methodenbenennung

Alle Daten, auf deren Basis Analysen gemacht werden, wie Programmcodes oder Programme die daraus resultieren, sind auf Produkte menschlicher Tätigkeiten zurückzuführen. Die korrekte Forschungsmethode, diese Inhalte zu analysieren und qualitativ zu erfassen, ist die Inhaltsanalyse. Das abschließende Testen der Interoperabilität hat einen experimentellen Charakter.
(vgl. Huber, 2005, S29)

4.2. Medien

Zum Thema „Web Services" gibt es ein breites Sortiment an Literatur. Durch den im Web angesiedelten Ursprung des Themas, befindet sich ein sehr großer Anteil an Schriftstücken im Internet wieder. Die Standards zu den aktuellen Web Service – Spezifikationen sind zum Beispiel vollständig im Web zu finden.

XML-RPC: http://www.xmlrpc.com/

SOAP: http://www.w3.org/

Zu den verwendeten Programmiersprachen gibt es ebenfalls eine Fülle von Dokumentationen und Handbüchern im Internet. Dennoch wurde davon abgesehen ausschließlich das Internet als Quelle zu nehmen, da sich Bücher vor allem für Sekundärliteraturbeschaffung gut eignen.

Die verwendeten Bücher und Internet Links sind im Literaturverzeichnis angeführt.

4.3. Untersuchungszeitraum

Für die Kommunikation werden die jeweils neuesten Programm- und Frameworkveröffentlichungen genommen, sofern sie im Rahmen dieser Diplomarbeit zugänglich sind. Der Zeitraum hierfür ist von 1. Jänner 2006 bis 31. Mai 2006, sofern geeignete Software in diesem Zeitraum aktualisiert zur Verfügung steht, andernfalls wird auf vorher hergestellte Software zurückgegriffen.

Geschichtliche Hintergründe gehen in den jeweiligen Teilbereichen dieser Diplomarbeit, aus Gründen des Umfangs der Thematik, nur so weit zurück, als das es das Verständnis erfordert.

5. Durchführung der Untersuchung

Die Untersuchung zur Beantwortung der Forschungsfragen wurde in den drei Programmiersprachen Java, PHP und C# (.NET), auf den jeweiligen Plattformen, durchgeführt. Hierbei haben sich vor allem technische Unterschiede in den unterschiedlichen Plattformen herauskristallisiert, welche bereits vor der Durchführung als Kategorien definiert wurden.

5.1. Kategorienbildung

Dimension: Technische Unterschiede

Kategorie: Erstellung von Web Services

Ausprägung: Toolunterstützung (ja/nein)
Ausprägung: Dokumentation (ausreichend/unzureichend)
Ausprägung: Integrationsaufwand (niedrig/hoch)

Kategorie: Interoperabilität der Web Services

Ausprägung: Interoperabilität mit anderen Implementierungen (ja/nein)

5.2. Testen der Interoperabilität

Im Rahmen der inhaltlichen Analyse werden die ausgewählten Implementierungen in das Testsystem integriert. Anschließend werden die Implementierungen gegeneinander getestet. Diese Tests obliegen ei-

nem definierten Ablauf und haben einen experimentellen Charakter, der im Folgenden beschrieben ist.

- Download der Implementierung
- Installation der Implementierung
- Test des Clients mit dem Web Service der Implementierung
- Tests des Clients mit anderen Web Services
- Tests anderer Clients mit dem Web Service der Implementierung

Die Übertragung wird bei jedem Test mit einem speziellen Programm überwacht. Dieses Programm nennt sich „Tunnel" und ist ein im „Apache SOAP" beigelegtes Tool, mit dem man die HTTP Daten zwischen Client und Server betrachten kann, indem man das Programm als Zwischenstation benutzt. Damit ist gemeint, dass man einen Port definiert (z.B.: 5555) und diesen bei den Clients als Web Service Port angibt. Anschließend wird der eigentliche Zielport des Web Service im Tool selbst definiert. Die Auswertung und Interpretation der Interoperabilität erfolgt abschließend im Kapitel 6.

5.3. Verwendete Hardware

Als Plattform für die Installationsversuche wurde ein Pentium 4 2,26GHz mit 1024MB DDR RAM ausgewählt.

5.4. Verwendete Software

Das Betriebsystem unter dem die Tests durchgeführt wurden ist Windows XP Professional mit Service Pack 2 (Student Edition). Die für die Erstellung eines Web Services benötigte plattformabhängige Software wird in den folgenden Unterkapiteln genauer beschrieben.

5.4.1. Entwicklungsumgebung für Java

Vorrangiges Ziel der Java Implementierung war es, ein Web Service gemäß der SOAP Spezifikation, als auch eines gemäß der XML-RPC Spezifikation zu erstellen und mit einem Java Client anzusprechen. Dazu musste zunächst einmal eine geeignete Testumgebung erstellt werden.

5.4.1.1. JRE und JDK

Java benötigt eine Virtual Machine auf dem Rechner um seine Programme ausführen zu können. Diese VM ist im Java Runtime Environment (JRE) Paket enthalten. Um aus Quelltext ausführbaren Bytecode erzeugen zu können, benötigt man das Java Development Kit (JDK), welches unter anderem den Java Compiler (javac.exe) enthält. Als JRE und JDK Version wurde 1.5.0 update 6 gewählt.

5.4.1.2. Web Server

Einer der gängigsten Web Server die im Java Bereich zum Einsatz kommen, ist der Tomcat Servlet-Container von Apache. Der Apache HTTP Server beinhaltet eine Referenzimplementierung der Spezifikation für Java Servlets und Java Server Pages (.jsp). Die offiziell eingesetzte Tomcat Version ist 5.5.17, obwohl die meisten Klassen auch mit der Version 5.0.28 getestet wurden. Beim Tomcat handelt es sich um Software, die der Apache License Version 2.0 unterliegt.

Der Web Server ist im Rahmen der Implementierungen dieser Arbeit auf Port 8180 festgelegt worden und somit mit http://localhost:8180/ anzusprechen.

5.4.1.3. Editor

Ein sehr guter und in der Literatur oft referenzierter Open Source Editor für Java ist Eclipse. Eigentlich handelt es sich dabei um eine Entwicklungsumgebung für größere Projekte, im Rahmen dieser Diplomarbeit wurde Eclipse jedoch ausschließlich als Editor verwendet. Wenn man tiefgehender in die Java Entwicklung von Web Services eintauchen möchte, empfiehlt es sich die WTP (Web Tool Plattform) Erweiterung für Eclipse zu installieren, bei der zahlreiche Hilfestellungen zur Web Service Erstellung unter Java mitgeliefert werden, wie zum Beispiel automatische Proxy Generierung.

5.4.2. Entwicklungsumgebung für .NET

Da in dieser Arbeit nur freie Software und Open Source Produkte zum Einsatz kommen sollen, wurde auf Visual Studio verzichtet. Es sei an dieser Stelle jedoch erwähnt, dass Visual Studio 2005 einige Web Service Erstellungshilfen bietet.

5.4.2.1. .NET Framework

Das auf dem Computer installierte .NET Framework trägt die Versionsnummer 2.0 und wird bei der Windows XP Installation nicht automatisch mitinstalliert. Zum Kompilieren benötigt man die csc.exe, die sich im .NET Paket befindet. Das Framework wird auch benötigt wenn man Anwendungen ausführen möchte, die mit dem .NET Framework entwickelt wurden.

5.4.2.2. Web Server

Als Webserver wurde der IIS (Internet Information Server) benutzt, der mit Windows XP mitgeliefert wird und standardmäßig auf http://localhost Port 80 läuft. Für die Dauer dieser Arbeit ist der Virtuelle Server auf dem Versuchsrechner unter http://localhost:84/ zu erreichen.

5.4.2.3. Editor

Als Editor wurde das Open Source Produkt SharpDevelop 2.0 benutzt, welches sich auf C# Programmierung spezialisiert, jedoch auch VB.NET unterstützt.

5.4.3. Entwicklungsumgebung für PHP

Da PHP vielseitig im Web Bereich Anwendung findet, darf es natürlich auch im Web Service Bereich nicht fehlen. Weil PHP in der Version 4 noch immer stark vertreten ist, wurde neben der Version 5 auch PHP4 auf dem Rechner installiert.

5.4.3.1. XAMPP

Als Entwicklungsumgebung wurde XAMPP von ApacheFriends, welches ebenfalls ein Open Source Produkt ist, eingesetzt. In der Version 1.5.1 sind die PHP Pakete 5.1.1 und 4.4.1-pl1 zusammen mit einigen Zusatztools enthalten. Gleichzeitig wird ein Windows Interface mitinstalliert, welches einfach zu bedienen ist. Man kann damit die einzelnen Dienste starten und stoppen, bzw. deren momentanen Status auch ausgeben lassen.

5.4.3.2. Web Server

Die Entscheidung des Web Servers ist für PHP auf den HTTP Server „Apache 2.2.0" gefallen, da es sich dabei um Open Source Software handelt und dieser bereits im XAMPP Paket enthalten ist. Sein Marktanteil von über 60% beim Darstellen von Webseiten, spricht ebenfalls für den Apache.
(vgl. http://news.netcraft.com/archives/web_server_survey.html, 05.06.2006)

5.4.3.3. Editor

Als Editor wurde ein einfacher Texteditor verwendet.

5.5. Integration von XML-RPC Implementierungen

Das Rad neu zu erfinden und den XML-RPC Standard selbst zu implementieren würde nicht nur den Rahmen dieser Arbeit bei weitem übersteigen, sondern auch eine vollkommen sinnlose Tätigkeit darstellen. Stattdessen wurden im Rahmen dieser Arbeit mehrere Implementierungen des XML-RPC Standards in den jeweiligen Programmierwelten näher betrachtet und getestet. Sämtliche URLs zu den Implementierungen sind im Literaturverzeichnis zu finden. Um die Übersichtlichkeit der Code Beispiele zu erhalten sind sowohl client- als auch serverseitig keinerlei Sicherheitsmechanismen implementiert, die ein unsachgemäßes Verwenden der Web Services behandeln, sofern diese nicht zur Lauffähigkeit des Programms erforderlich sind.

5.5.1. XML-RPC Implementierungen für Java

5.5.1.1. Apache XML-RPC

Die XML-RPC Implementierung in Java von Apache ist in der Literatur sehr oft zitiert. Xmlrpc-3.0a1 heißt die verwendete Bibliothek, die wiederum einige Abhängigkeiten an andere Javabibliotheken stellt. Zu den Anforderungen zählen die Pakete „org.apache.ws.commons", "xerces" und "javax.servlet", wobei das ws.commons Paket nicht einfach zu finden ist.

Apache XML-RPC Server Implementierung in Java

Um ein XML-RPC Web Service mit der Apache Implementierung zu erstellen schreibt man zunächst eine Java Klasse, welche die Methoden beinhaltet, die man später anbieten möchte.

```
01  public class JavaXmlrpcApacheServer{
02
03      public int add(int i1, int i2) {
04          return i1 + i2;
05      }
06      public int subtract(int i1, int i2) {
07          return i1 - i2;
08      }
09      public int multiply(int i1, int i2) {
10          return i1 * i2;
11      }
12      public double divide(int i1, int i2) {
13          return (double)i1 / (double)i2;
14      }
15
16  }
```

Code 05 – JavaXmlrpcApacheServer.java

Damit der Apache Web Server diese erstellte Klasse finden kann, muss eine „.properties" Datei im Ordner „tomcat\webapps\projektname\

WEB-INF\classes\org\apache\xmlrpc\webserver\" angelegt werden, die den Titel „XmlRPcServlet.properties" trägt. In dieser Datei stehen lediglich der Name des Endpunktes auf der linken und der Name der Klasse auf der rechten Seite.

```
Math = JavaXmlrpcApacheServer
```

Gehört die erstellte Klasse zu einem „Package", so muss dieses auf der rechten Seite in Java üblicher Punktnotation angegeben werden.

```
Math = package.JavaXmlrpcApacheServer
```

Der letzte Schritt ist das Schreiben einer web.xml Datei, die der Tomcat benötigt um das Service wie ein Servlet aufrufen zu können. Das Erstellen der web.xml wird von diesem Paket leider nicht automatisiert durchgeführt, man muss die Datei manuell im zuvor bereits erwähnten „WEB-INF" Ordner anlegen.

```
01  <?xml version="1.0" encoding="ISO-8859-1"?>
02  <!DOCTYPE web-app PUBLIC "-//Sun Microsystems, Inc.//DTD
    Web Application 2.3//EN" "http://java.sun.com/dtd/web-
    app_2_3.dtd">
03  <web-app>
04      <display-name>XML-RPC Service</display-name>
05      <description>Web Service Integration with XML-
    RPC</description>
06      <servlet>
07          <servlet-name>XmlRpcServlet</servlet-name>
08          <servlet-
    class>org.apache.xmlrpc.webserver.XmlRpcServlet</servlet
    -class>
09      </servlet>
10      <servlet-mapping>
11          <servlet-name>XmlRpcServlet</servlet-name>
12          <url-pattern>/xmlrpc</url-pattern>
13      </servlet-mapping>
14  </web-app>
```

Code 06 – XML-RPC Apache web.xml

In Zeile 07 und Zeile 11 steht der Name des Servlets, der in beiden Zeilen identisch sein muss. In Zeile 08 sieht man den Verweis auf die properties Datei mit der auch der Tomcat die erstellte Klasse findet. Beim Mapping (Zeilen 10-13) wird die URI definiert unter der das Web Service erreichbar ist. Die vollständige Adresse lautet in diesem Fall:

http://localhost:8180/web/xmlrpc wobei „web" der Projektname ist, in dem sich der WEB-INF Ordner befindet.

Nach einem Neustart vom Tomcat ist das XML-RPC Web Service ansprechbar.

Apache XML-RPC Client Implementierung in Java

Der vollständige Code des Clients, in dem alle Methoden des Web Services aufgerufen werden, ist auf der beiliegenden CD-ROM zu finden. Hier wird lediglich eine Methode aufgerufen, daher ist der Code zu Demonstrationszwecken leicht modifiziert.

```
/* Die folgenden Bibliotheken sind notwendig */
import org.apache.xmlrpc.client.XmlRpcClient;
import org.apache.xmlrpc.client.XmlRpcClientConfigImpl;
import org.apache.xmlrpc.XmlRpcException;
import java.net.URL;


/* Die URL des Web Service muss in einem URL Objekt gespeichert werden
bevor sie im Konfigurationsobjekt (config) gespeichert wird */
XmlRpcClientConfigImpl config = new
XmlRpcClientConfigImpl();

URL urlobj = new URL("http://localhost:8180/web/xmlrpc");
config.setServerURL(urlobj);

/* Erstellung des Client Objekts und Zuweisung der Config Datei */
XmlRpcClient client = new XmlRpcClient();
client.setConfig(config);
```

```
/* Der eigentliche Aufruf geschieht mit der Methode client.execute die
als ersten Parameter den Endpunktnamen, gefolgt von dem Namen der aufzu-
rufenden Methode haben muss und als zweiten Parameter die Übergabewerte
in einem Object mit gesandt bekommt */
Object[] params = new Object[]{3,5};
result = client.execute("JavaXmlrpcApacheServer.add“,
params).toString();
```

5.5.1.2. RoX

Während das Apache XML-RPC Paket lediglich mit der Java Version 1.5 reibungslos funktioniert, hat der Entwickler James Greenfield eine Implementierung geschaffen, die auch mit der Java Umgebung 1.4 funktioniert. Die Servervariante die diese Implementierung zu Verfügung stellt, erstellt ihren eigenen Web Server.

Rox Server Implementierung in Java

Der Server wird über die Konsole gestartet und läuft, solange die Konsole geöffnet ist, auf einem im Quellcode angegebenen Port.

```
/* Zunächst benötigt diese Implementierung eine Reihe von Bibliotheken
*/
import java.net.*;
import java.util.*;
import java.io.*;
import com.flat502.rox.server.*;
import com.flat502.rox.marshal.*;
import com.flat502.rox.marshal.xmlrpc.*;

/* Da dieser Ansatz von den Web Server nutzenden Implementierungen ab-
weicht wird ein Großteil der Serverklasse gezeigt */
public class JavaXmlrpcRoxServer implements
SyncRequestHandler {
    public RpcResponse handleRequest(RpcCall call)
    throws Exception {

/* Die Parameter werden aus dem Call Objekt, welches dem Handler überge-
ben wird ausgelesen und in das „Object[]“ Objekt params gespeichert.
Durch die Verwendung von Objekt können beliebige Werte übergeben werden
*/
        Object[] params = call.getParameters();
        Integer result = 0;
```

```
/* Der Methodenaufruf von add geschied erst wenn der Name validiert wur-
de. Erst dann werden die Parameter an die Methode add Übergeben */
        if (call.getName().equals("Math.add")) {
            result = (Integer)add((Integer) params[0],
(Integer) params[1]);


/* An dieser Stelle wird das Ergebnis retourniert */
            return new XmlRpcMethodResponse(result);
        }
    }

/* Hier wird definiert was in der add Methode passieren soll */
    private int add(Integer x, Integer y) {
        return (Integer) (x + y);
    }

/* In der „main“ Methode wird das Serverobjekt erzeugt */
    public static void main(String[] args) {
        String host = "localhost";
        int port = 8181;

/* Dem Serverobjekt werden Pfad und Port übergeben unter denen der Ser-
ver erreichbar sein soll */
        XmlRpcServer server = new XmlRpcServer(
InetAddress.getByName(host), port);

/* Der Handler ist ein Objekt dieser Klasse, er benötigt drei Parameter,
den URI Pfad (null steht für alle), den Namensraum im Format
^Namensraum\\ und eine Instanz dieser Klasse */
        server.registerHandler(
„http://localhost:8181/xmlrpc“, "^Math\\.", new
JavaXmlrpcRoxServer());
        server.start();

    }
}
```

Der Vorteil eines solchen Servers liegt auf der Hand, man benötigt keinen vorinstallierten Web Server zum Anbieten von Web Services.

Rox Client Implementierung in Java

```
/* Zum Unterschied zur Apache Implementierung wird hier kein URL Objekt
benötigt, die URL kann direkt als String übergeben werden */
String url = "http://localhost:8180/web/xmlrpc";
```

```
/* Die Parameter werden erneut als Objekt Übergeben */
Object[] params = new Object[]{4,5};
XmlRpcClient client = new XmlRpcClient(new URL(url));

/* Der eigentliche Aufruf geschieht auf gleiche Weise wie in der Apache
Implementierung */
Object result = client.execute("Math.add", params);
```

5.5.2. XML-RPC Implementierungen für .NET

5.5.2.1. XML-RPC.NET

Sucht man nach XML-RPC Implementierungen, so findet man auf vielen Wegen zu dieser. Die zum Zeitpunkt der Durchführung dieser Arbeit aktuelle Version 0.9.2 unterstützt bedauernswerter Weise nicht das Framework 2.0. Dieses wird erst mit der Version 1.0.0 unterstützt, die am 12.7.2006 erscheint.

5.5.2.2. XmlRpcCS

Ein sehr interessanter Ansatz für einen XML-RPC Server bzw. Client ist mit der Bibliothek XmlRpcCS umgesetzt worden. Mit dieser Implementierung kann man Web Services erzeugen die unabhängig vom IIS laufen. D.h., ähnlich wie bei RoX, spezifiziert man einen Port und startet den Server, der standardmäßig auf Port 5050 eingestellt ist. Literatur gibt es nur spärlich zu diesem Produkt, da die Web Sites, die auf die Dokumentation verweisen, nicht mehr existieren.

Dennoch konnte anhand der Beispiele die im Paket enthalten sind ein funktionierender Server bzw. Client erzeugt werden.

XmlRpcCS Server Implementierung in .NET

Der vollständige Quellcode ist auf der beiliegenden CD-ROM zu finden.

```
namespace Math

/* Zunächst wird der Port festgelegt auf dem des Web Service erreichbar
sein soll */
const int PORT = 5050;

/* Das Server Objekt wird erzeugt und bekommt eine Instanz der
NetXmlrpcCSServer Klasse hinzugefügt (Math ist der Namespace). Anschlie-
ßend wird er gestartet */
XmlRpcServer server = new XmlRpcServer(PORT);
server.Add("Math", new NetXmlrpcCSServer());
server.Start();

/* Die angebotenen Methoden werden nach der main Methode (siehe CD-ROM)
am Ende der Klasse definiert */
public int add(int x, int y)
{
    return x + y;
}
```

XmlRpcCS Client Implementierung in .NET

```
/* Zunächst erstellt man ein Response Objekt und den XML-RPC Client */
XmlRpcResponse response;
XmlRpcRequest client = new XmlRpcRequest();

/* Der Methodenname und die Parameter werden dem Client Objekt hinzuge-
fügt und mit Client.Send(„WebServiceAdresse“) abgeschickt */
client.MethodName = "Math.add";
client.Params.Clear();
client.Params.Add(5);
client.Params.Add(7);
response = client.Send("http://localhost:5050");
Console.WriteLine("Response: " + response.Value);
```

5.5.3. XML-RPC Implementierungen für PHP

5.5.3.1. XML-RPC for PHP 2.0

Diese Open Source XML-RPC Implementierung für PHP ist von dem Editor des O’Reilly Netzwerks (http://www.oraillynet.com), Edd Dumbill, ins Leben gerufen worden und gilt als sehr umfangreiche XML-RPC Implementierung, an der sich im Laufe der Zeit eine Menge Leute be-

teiligt haben. Durch ihre Komplexität ist sie jedoch nicht die schnellste oder Ressourcen sparendste Implementierung für PHP, da sowohl beim Client als auch beim Server einige Umwandlungen in diverse Objekte notwendig sind (siehe folgender Sourcecode). Die Mindestanforderung für die PHP Version ist 4.2.

(vgl. http://phpxmlrpc.sourceforge.net/, 28.07.2006)

XML-RPC for PHP 2.0 Server Implementierung

```
/* Diese Pakete werden im Package mitgeliefert */
include("lib_xmlrpc20/lib/xmlrpc.inc");
include("lib_xmlrpc20/lib/xmlrpcs.inc");

/* Die Implementierung benötigt, anders als die bisherigen Server Imple-
mentierungen, die zwei Integer als Parameter hatten, ein Message Objekt
als Input Parameter */
function add ($xmlrpcmsg) {

/* Zunächst muss man die xmlrpcval Parameter aus dem xmlrpcmsg Objekt
holen (man kann sich die Parameteranzahl auch in einer Schleife ausgeben
lassen) */
    $par1 = $xmlrpcmsg->getParam(0);
    $par2 = $xmlrpcmsg->getParam(1);

/* Um den Integer Wert der xmlrpcval Objekte zu erhalten, benötigt man
die Methode scalarval */
    $result = $par1->scalarval() + $par2->scalarval();

/* Bevor der Wert an den Client zurückgesendet wird, muss er erneut in
ein xmlrpcval Objekt eingefügt werden */
    return new xmlrpcresp(new xmlrpcval($result, "int"));
}

/* Die Methode add() muss beim Server Objekt als Service Methode regist-
riert werden */
$s = new xmlrpc_server(
    array("Math.add" => array("function" => "add"),)
);

/* Die Service Methode behandelt ankommende Aufrufe */
$s->service();
```

XML-RPC for PHP 2.0 Client Implementierung

```
/* Für den Client ist lediglich die xmlrpc.inc Datei notwendig */
include("lib_xmlrpc20/lib/xmlrpc.inc");

/* Typisch für eine XML-RPC Implementierung wird zunächst ein Client Objekt erzeugt, dem die Zielserver URL übergeben wird */
$server_url = "http://localhost:81/php_projects/XML-RPC/PhpXmlrpcIXRServer.php";
$client = new xmlrpc_client($server_url);

/* Es muss ein xmlrpc_message Objekt erstellt werden, welches den Methodennamen und die Parameter als Array erhält. Die Parameter müssen dem Array einzeln als xmlrpcval Objekte hinzugefügt werden */
$methodName = "Math.add";
$parameterArray = array(new xmlrpcval(23, "int"),new xmlrpcval(13, "int"));
$xmlrpc_message = new xmlrpcmsg($methodName, $parameterArray);

/* Der eigentliche Web Service Aufruf erfolgt hier */
$xmlrpcresponse = $client->send($xmlrpc_message );

/* Zum Auslesen des Integer Wertes muss das xmlrpcresponse Objekt zunächst den Value angeben, aus dem wiederum die Scalarvariable (der Integer Wert) ausgelesen werden muss */
$val = $xmlrpcresponse->value();
echo $methodName . ": " . $val->scalarval() . "<br>";
```

Diese Art der Implementierung hat den Vorteil, dass man sehr kompetente Clients schreiben kann, die auf viele verschiedene Rückgabeparameter reagieren können. Grund dafür ist, dass man diverse Methoden zur Verfügung hat um herauszufinden was in einem „xmlrpcresponse" Objekt alles beinhaltet ist.

5.5.3.2. IXR

Diese Implementierung wurde ausgewählt weil sie zum einen sehr performant und zum anderen sehr schnell angewendet ist. Der vollständige Code von Client und Server ist auf der beiliegenden CD-ROM nachzulesen.

IXR Server Implementierung für PHP

Die Methoden des Servers bekommen unabhängig von Struktur und Typen der gesendeten Parameter immer ein Array übergeben. Das bedeutet: Unabhängig davon ob man dem Web Service Aufruf ein Array in dem zwei Integer enthalten sind, oder zwei getrennte Integer übergibt, werden die Integer durch die IXR Implementierung in ein Array umgewandelt. Die Methoden dürfen in der Server Klasse also nicht mit mehr als einem Eingangsparameter definiert werden. Der Nachteil der sich daraus ergibt ist offensichtlich, es kann aus der Klasse keine WSDL Datei automatisch generiert werden (bzw. nicht ohne extremen Programmieraufwand), da die Übergabeparameter der Methoden nicht transparent sind.

```
/* In dieser Datei befindet sich die gesamte IXR Implementierung */
include("lib/IXR_Library.inc.php");

/* Hier ist ein Beispiel für eine Methode, es darf nur ein Parameter
übergeben werden, wie dieser heißt ist jedoch irrelevant */
function add($args) {
    return $args[0] + $args[1];
}

/* Erstellung des Server Objekts und Mappen der XML-RPC Methodennamen zu
den relevanten Funktionen, „Math“ simuliert hier den Namensraum */
$server = new IXR_Server(array("Math.add" => "add"));
```

IXR Client Implementierung für PHP

```
/* Auch für den Client reicht die Einbindung einer Datei */
include("lib/IXR_Library.inc.php");

/* Erzeugung des Clients mit Übergabe der URL des aufzurufenden Web Service */
$client = new IXR_Client ("http://locahost:81/php_projects/
XML-RPC/PhpXmlrpcIXRServer.php");

/* Der eigentliche Web Service Aufruf erfolgt hier */
$client->query("Math.add", 5, 8);
echo $client->getResponse();
```

Man sieht auf den ersten Blick, dass diese Implementierung sehr simpel gehalten ist und sowohl Client als auch Server in sehr kurzer Zeit erstellt werden können. Trotz der einfachen Bedienung unterstützt sie alle für XML-RPC übliche Typen und auch verschachtelte Parameter sind möglich.

IXR Advanced Client Implementierung für PHP

Besonders an IXR ist, dass es multiple Funktionsaufrufe mit einmaligem Senden der HTTP Anfrage unterstützt:

```
/* Erzeugung eines Multicall Clients */
$client = new
IXR_ClientMulticall('http://localhost:81/php_projects/XML-
RPC/PhpXmlrpcIXRServer.php');

/* Definition der Zahlen */
$x = 5; $y = 8;

/* Die verschiedenen Methoden die aufgerufen werden sollen werden dem
Client Objekt mit addCall(Methode, $parameter1, &parameter2,…) hinzuge-
fügt */
$client->addCall('Math.add', $x, $y);
$client->addCall('Math.subtract', $x, $y);
$client->addCall('Math.multiply', $x, $y);
$client->addCall('Math.divide', $x, $y);

/* Hier erfolgt der Aufruf des Web Service (es wird tatsächlich nur eine
HTTP Verbindung geöffnet) */
$client->query();

/* Zurückgeliefert werden 4 Arrays die in einem Überarray enthalten sind
und jeweils die Rückgabewerte enthalten */
$response = $client->getResponse();
for ($i = 0; $i < count($response); $i++)
    echo $response[$i][0] . "<br>";
```

Das Ergebnis wird vom Server in Arrays geschachtelt und anschließend über eine (nicht vier aufeinander folgende) Verbindungen übertragen. Das erklärt auch warum das $response Objekt die Arrays enthält.

5.6. Integration von SOAP Implementierungen

Wie schon beim XML-RPC Standard wurden auch beim SOAP Standard mehrere Implementierungen auf den verschiedenen Plattformen getestet. Die Auswahl der jeweiligen Implementierungen erfolgte – sofern nicht anders im Detail angegeben – durch umfassende Literaturrecherchen. Gesucht wurde nach einer leichten Umsetzbarkeit und einem oftmaligen Einsatz der Implementierung. Untersucht wurden die Implementierungen auf problemlosen Einsatz, Tool Unterstützungen und Funktionalität. Die Interoperabilität ist in Kapitel 6 beschrieben. Die Quellen der jeweiligen Implementierungen sind im Literaturverzeichnis angeführt. Im Rahmen des Tests werden bei allen Web Services die Funktionen Addieren, Subtrahieren, Multiplizieren und Dividieren angeboten, jedoch werden, im Sinne der einfacheren Lesbarkeit, Funktionen bei der Darstellung der Clients weggelassen, wenn sie für das Verständnis nicht erforderlich sind.

5.6.1. SOAP Implementierungen für Java

Es gibt zahlreiche Implementierungen des SOAP Standards für Java, die als Open Source Software verfügbar sind.

5.6.1.1. Apache SOAP

Für die Entwicklung des SOAP Servers und Clients wurde als erste Implementierung Apache SOAP v2.3.1 gewählt, da Apache einerseits Open Source Produkte bereit stellt und das SOAP Paket in der Literatur als sehr gute SOAP Implementierung gilt. Es gibt jedoch einige Abhängigkeiten die dieses Paket benötigt, ohne denen es nicht möglich ist, die zur Verfügung gestellten Klassen zu nutzen. Diese Abhängigkeiten sind Java in der Version 1.2 oder höher, JavaMail (mail.jar), JavaBeans Ac-

tivation Framework (activation.jar) und XMI encoding (xerces.jar und xml4j.jar).

Apache SOAP Server Implementierung für Java

Begonnen wird mit der Erstellung der eigentlichen Rechenklasse, in der die Methoden, die das Web Service anbieten soll, definiert werden.

```
01  public class JavaSoapApacheServer {
02
03      public int add(int p1, int p2) {
04          return p1 + p2;
05      }
06
07      public int subtract(int p1, int p2) {
08          return p1 - p2;
09      }
10
11      public int multiply(int p1, int p2) {
12          return p1 * p2;
13      }
14
15      public double divide(int p1, int p2) {
16          return (double)p1 / (double)p2;
17      }
18
19  }
```

Code 07 – JavaSoapApacheServer.java

Anschließend muss die Klasse auf dem Web Server eingebunden und damit nach Außen hin nutzbar gemacht werden. Eine detaillierte Beschreibung, wie das von statten geht, ist unter der Adresse http://www.soapuser.com/server1.html zu finden, es seien hier aber die wichtigsten Punkte erwähnt.

Um als Web Service angeboten zu werden, muss die erstellte Klasse, mit dem „Deployment Descriptor", nach Außen bekannt gemacht wer-

den. Dieser wird als Datei „deploymentDescriptor.xml“ im /WEB-INF/ Verzeichnis der Web Applikation abgespeichert.

```
01  <isd:service xmlns:isd="http://xml.apache.org/xml-
    soap/deployment"
02      id="urn:soapserver">
03          <isd:provider type="java"
04              scope="Application"
05              methods="add subtract multiply divide">
06              <isd:java class="JavaSoapApacheServer"/>
07          </isd:provider>
08
    <isd:faultListener>org.apache.soap.server.DOMFaultListen
    er</isd:faultListener>
09  </isd:service>
```

Code 08 – Apache Soap deploymentDescriptor.xml

Es ist wichtig, dass die URN in Zeile 2 eindeutig ist. Man kann also nicht zwei gleich lautende Identifikationsbezeichnungen in einer Webapplikation vergeben. Zeile 6 zeigt auf die verwendete Klasse. Zusätzlich kann man noch die Option <isd:mappings> hinzufügen um dort komplexe Datentypen zu definieren, was hier allerdings nicht erforderlich ist. Die Datei web.xml, in der sich die URI Information befindet, welche definiert unter welcher Adresse das Web Service später erreichbar ist, ist im SOAP-Paket bereits enthalten und muss sich ebenfalls im /WEB-INF/ Verzeichnis befinden, was jedoch beim Einbinden durch den Deployment Descpriptor automatisch passiert. Die URI, die in der web.xml in <servlet-mapping> standardmäßig als „/servlet/rpcrouter“ definiert ist, kann an dieser Stelle verändert werden. Der Aufruf des Web Service erfolgt mit den Standard Einstellungen auf folgende Weise:

http://localhost:8180/soap/servlet/rpcrouter

Wobei „8180“ in der Testumgebung der Port des Tomcats ist und „soap“ für das Projektverzeichnis steht.

Der Deployment Descriptor wird vom Tomcat nicht automatisch eingebunden. Wie diese Einbindung im Detail funktioniert kann unter http://www.soapuser.com/server3.html nachgelesen werden.

Apache SOAP Client Implementierung für Java

Beim Kompilieren der Client Klasse wird die Library soap.jar aus dem Apache Paket benötigt. Weiter braucht man zur Ausführung die mail.jar und die activation.jar im classpath. Der vollständige Code ist auf der beiliegenden CD-ROM zu finden, im Folgenden werden nur Teile des Codes zur Erklärung aufgelistet.

```
/* Zunächst definiert man die Web Server URL als URL-Object */
URL url = new URL ("http://localhost:8180/soap/servlet/
rpcrouter");

/* Die Parameter müssen bei dieser Implementierung in ein Objekt des
Typs „Vector“ geschachtelt werden */
Vector params   = new Vector();
params.addElement(new Parameter("x", Integer.class,
New Integer(3), null));
params.addElement(new Parameter("y", Integer.class,
New Integer(3), null));

/* Das Call Objekt ist dafür zuständig das Web Service aufzurufen. In-
formationen wie die Web Service Bezeichnung (URN), die Enkodierungsvari-
ante, der Methodenname und die mitzuschickenden Parameter werden dem
Call Objekt zugewiesen */
Call call = new Call();
call.setTargetObjectURI("urn:soapserver");
call.setEncodingStyleURI(Constants.NS_URI_SOAP_ENC);
call.setParams (params);
call.setMethodName(“add”);

/* Hier erfolgt der eigentliche Aufruf vom Web Service, mit anschließen-
der Ausgabe des Resultats */
Response resp = call.invoke(url, "" );
result = resp.getReturnValue();
System.out.println(result.getValue());
```

Ein sehr großes Manko dieser Implementierung ist, dass es keine WSDL Unterstützung gibt. Die Metainformationen der Aufrufe, die normaler-

weise aus einer WSDL Datei ausgelesen werden, müssen manuell hinzugefügt werden.

5.6.1.2. Apache Axis

Ein weiteres Open Source SOAP Produkt von Apache ist Apache Axis. Durch seine Tool Unterstützung zur Erstellung von Web Services bietet Axis im Vergleich zu Apache SOAP einen verhältnismäßig großen Funktionsumfang an. Unter anderem werden den Usern Tools wie WSDL2Java und Java2WSDL, mit denen, wie der Name schon sagt, WSDL Dateien in Javacode (bzw. umgekehrt) umgewandelt werden können, angeboten. Weiters kann man mit Axis SOAP Web Services mittels WSDL Unterstützung aufrufen.

Auch Axis hat einige Systemanforderungen, die nach der Installation des Pakets unter http://localhost:8180/axis/happyaxis.jsp überprüft und angezeigt werden.

Axis Easy SOAP Server Implementierung für Java

Axis stellt zwei Möglichkeiten bereit eine Klasse als Web Service anzubieten. Die erste Möglichkeit ist die einfachere aber weniger flexible Variante. Weniger flexibel deshalb, weil man kaum Einfluss auf das Deployment, die Art der Einbindung und die Auswahl der zu exportierenden Methoden hat.

(vgl. http://www.torsten-horn.de/techdocs/java-soap-axis.htm#Installation-ApacheAxis, 30.07.2006)

Für die Implementierung ist es lediglich erforderlich, die Methoden welche angeboten werden sollen, in einer Klasse zu definieren

```
01  public class Math {
02      public int add (int a, int b) {
03          return a + b;
04      }
05          public int subtract (int a, int b) {
06          return a - b;
07      }
08          public int multiply (int a, int b) {
09          return a * b;
10      }
11          public double divide (int a, int b) {
12          return a / b;
13      }
14  }
```

Code 09 – Math.jws

Diese .jws Datei kopiert man anschließend in das Installationsverzeichnis von Axis auf dem Web Server und startet diesen neu. Nun kann man das Web Service folgendermaßen ansprechen:

http://localhost:8180/axis/Math.jws

Mit dem Parameter „?wsdl" kann man auch die automatisch erstellte WSDL Datei betrachten.

Axis Advanced SOAP Server Implementierung für Java

Die „Deployment" Prozedur entspricht im Großen und Ganzen der Apache XML-RPC Server Implementierung und kann unter http://www.torsten-horn.de/techdocs/java-soap-axis.htm#Installation-ApacheAxis im Detail nachgelesen werden.

Kurz zusammengefasst erstellt man zunächst die Klasse mit den Methoden, die man anbieten möchte (z.B. die gerade erstellte Math.jws Datei, allerdings diesmal mit .java Endung). Außerdem benötigt man einen „Deployment Descriptor" der in das /WEB-INF/ Verzeichnis der

Applikation kopiert und anschließend ausgeführt werden muss. Der Deployment Descriptor ist eine XML Datei in der angegeben wird, wie die Klasse heißt die man einbinden möchte (eventuell auch mit Package Name). Außerdem können Zusatzparameter, wie „allowedMethods" (man kann hier Methoden ausschließen) und „scope" (Bereich der Gültigkeit), definiert werden.

```
01  <deployment xmlns="http://xml.apache.org/axis/wsdd/"
02  xmlns:java="http://xml.apache.org/axis/wsdd/providers/
    java">
03    <service name="MeinWebService" provider="java:RPC">
04      <parameter name="className"
    value="meinpackage.MeineWebServiceKlasse2" />
05      <parameter name="allowedMethods" value="*" />
06      <parameter name="scope" value="Request" />
07    </service>
08  </deployment>
```

Code 10 – Deploy.wsdd

Durch die Möglichkeit Methoden auszuschließen oder den Gültigkeitsbereich einzuschränken hat man durch diese Art der Einbindung viel mehr Kontrolle über das Web Service als bei der simpleren Einbindungsart.

Anzusprechen ist das Web Service unter:

http://localhost:8180/axis/services/SoapServer

Axis SOAP Client Implementierung für Java ohne WSDL

Axis bietet die Möglichkeit WSDL Dateien zu nutzen oder Web Services direkt anzusprechen.

```
/* Zunächst wird die Server URI als Endpunkt definiert sowie der Namens-
raum und die aufzurufende Methode angegeben */
String wsEndpoint =
"http://localhost:8180/axis/services/SoapServer";
```

```
String wsNamespace = "SoapServerSoap";
String wsMethod = "add";

/* Die Parameter müssen in ein Objekt für die Übertragung gespeichert
werden */
Object[] val = new Object[2];
val[0] = new Integer( 8 );
val[1] = new Integer( 5 );

/* Bei dieser Implementierung wird zunächst ein Service Objekt angelegt,
das wiederum ein Call Objekt erzeugt */
Service service = new Service();
Call call = (Call)service.createCall();


/* Dem Call Objekt werden nun die oben definierten Parameter übergeben
bevor es mit call.invoke(parameter) das Web Service aufruft */
call.setTargetEndpointAddress( new URL( wsEndpoint ) );
call.setOperationName( new javax.xml.namespace.QName(
wsNamespace, wsMethod ) );
Object ret = call.invoke( val );

/* Zurückgegeben wird ein Objekt des Typs „Object“ */
System.out.println( ret );
```

Axis SOAP Client Implementierung für Java mit WSDL

Die Benutzung von WSDL Dateien hat den Vorteil, dass man die Parameter nicht manuell an das Call Objekt übergeben muss. Um eine WSDL Datei eines Web Services zu nutzen benötigt man das Tool WSDL2Java, welches im Axis Paket enthalten ist und das einen Proxy aus der zur Verfügung stehenden WSDL Datei generiert. Mit folgender Kommandozeile wird aus den Inhalten der WSDL Datei ein Package erstellt, das als Proxy fungiert, mit dem man das Web Service wie ein lokales Objekt nutzen kann.

```
java org.apache.axis.wsdl.WSDL2Java -o src -p
SoapServerWSDL2Java
http://localhost:8180/axis/services/SoapServer?wsdl
```

Der Parmeter –o gibt das Ausgabeverzeichnis an, während der Parameter –p den Package Namen angibt. Das erstellte Package muss vor der Verwendung noch kompiliert werden.

```
/* Zunächst wird im Client das kompilierte Package, sowie die Axis SOAP
Bibliothek eingebunden */
import SoapServerWSDL2Java.*;
import javax.xml.rpc.*;

/* Anschließend wird ein neues Service aus der Proxyklasse erstellt */
JavaSoapServerService service =
new JavaSoapServerServiceLocator();

/* In der JavaSoapServer Klasse sind die Methoden definiert, die im
„stub“ abgebildet werden. */
SoapServerWSDL2Java.JavaSoapServer stub =
service.getSoapServer();

/* Der Stub lässt sich wie ein lokales Objekt behandeln */
System.out.println( "Add: " + stub.add( x, y ) );
```

Durch das Benützen von WSDL Dateien lässt sich der Einbindungsvorgang dieses Clients weitgehend automatisieren, es ist daher empfehlenswert, sofern eine WSDL Datei vorhanden ist, diese auch zu benutzen.

5.6.2. SOAP Implementierungen für .NET

5.6.2.1. .NET Framework SOAP Web Services

Das .NET Framework liefert die Grundbausteine für die Web Service Entwicklung gleich mit, daher werden keine anderen externen Implementierungen benötigt. Da es sich bei den bisherigen Implementierungen jeweils um Webanwendungen handelt, stützt sich dieses Kapitel auf ASP.NET Technologien.

SOAP Server Implementierung in .NET

Zunächst erstellt man eine .asmx Datei, in der man die Web Methoden mit dem so genannten „Custom Attribut“ namens „[WebMethod]“ definiert. Methoden die nicht mit diesem Attribut gekennzeichnet sind,

werden, auch wenn sie als „public“ definiert sind, nicht vom Web Service zur Verfügung gestellt.

(vgl. Duthie, MacDonald, 2003, [eBook] Kapitel 30.1)

```
01  <%@ WebService Language="C#" Class="NetSoapServer" %>
02  using System;
03  using System.Web.Services;
04
05  public class NetSoapServer : WebService
06  {
07      [WebMethod]
08      public int add(int a, int b)
09      {
10          return a + b;
11      }
12      [WebMethod]
13      public int subtract(int a, int b)
14      {
15          return a - b;
16      }
17      [WebMethod]
18      public int multiply(int a, int b)
19      {
20          return a * b;
21      }
22      [WebMethod]
23      public double subtract(int a, int b)
24      {
25          return Convert.ToDouble(a)/Convert.ToDouble(b);
26      }
27  }
```

Code 11 – NetSoapServer.asmx

Dieses Service lässt sich anschließend gleich testen wenn man es in einem virtuellen Verzeichnis (hier „sharp/SoapServer“) des IIS abgelegt hat. Die Technologie die es ermöglicht, Assemblies (Bausteine einer .NET Framework Anwendung) während der Laufzeit zu untersuchen, heißt „Reflektion“. Mittels Reflektion kann man Assemblies auch dynamisch zur Laufzeit laden und binden.

(vgl. Weyer, 2002, S71)

Erreichbar ist das Service somit unter folgender Adresse:

http://localhost:84/sharp/soapServer/NetSoapServer.asmx

Die WSDL Datei wird ebenso mittels Reflektion erstellt, indem die mit den Attributen [WebMethod] gekennzeichneten Methoden untersucht und ausgelesen werden. Aufgerufen werden kann die WSDL unter der Service URL und dem angehängten Parameter „?wsdl".

SOAP Client Implementierung in .NET

Zunächst ist es notwendig einen Proxy zu erstellen, der für den Zugriff auf das Web Service zuständig ist. Will man den Proxy nicht von Hand schreiben, empfiehlt es sich die wsdl.exe zu benutzen, die im .NET Framework mitgeliefert wird. Die Datei wsdl.exe interpretiert die Web Service Infrastruktur durch das WSDL File und generiert anhand der Methoden und Parameter die dort formuliert sind eine Proxy Klasse, die das Interagieren mit dem Zielserver möglich macht.
(vgl. http://www.microsoft.com/germany/msdn/library/web/ WebServicesMitNETFramework20UndVisualStudio2005.mspx?mfr=true, 30.07.2006)

Mit der folgenden Kommandozeile wird ein Proxy auf dem Client erstellt:

```
wsdl /language:cs  /protocol:soap /namespace:netSoapProxy
/out:netSoapProxy.cs http://localhost:84/sharp/soapServer/
netSoapServer.asmx
```

Das Resultat ist eine netSoapProxy.cs Datei, die anschließend durch den Befehl „csc.exe /target:library netSoapProxy.cs" in eine dll umgewandelt wird. Diese .dll Datei muss bei ASP.NET Web Anwendungen direkt in das „bin" Verzeichnis im virtuellen Verzeichnis kopiert werden. Im aktuellen Beispiel also in „http://localhost:84/sharp/bin".

Im Folgenden sind die wichtigsten Code Zeilen beschrieben die ein ASP Client beinhalten muss.

```
/* Zunächst wird der Namensraum deklariert und bestimmt, dass das ASP
Script serverseitig läuft */
<% @Import Namespace="netSoapProxy" %>
<script language="C#" runat="server">

/* Diese Konvertierung ist ASP.NET bezogen und soll demonstrieren, dass
die Typen denen entsprechen müssen, die das WSDL vorgibt. a und b stehen
für die zwei Integer Werte die addiert werden */
int aint = Convert.ToInt32(a.Text);
int bint = Convert.ToInt32(b.Text);

/* Der eigentliche Aufruf besteht lediglich aus der Generierung eines
Objekts des Proxies, der gerade erstellt wurde und aus dem Aufruf der
Methode. Zu bemerken ist, dass hier wie bei der Axis Implementierung das
Proxy Objekt direkt mit den Methodennamen agieren kann. */
NetSoapServer myService = new NetSoapServer();

/* lblOut ist der Name eines ASP.NET Textfeldes */
lblOut.Text = myService.add(aint, bint).ToString();
```

Der eigentliche Client ist sehr kurz gehalten, die meiste Arbeit übernimmt die oben beschriebene wsdl.exe, die die Proxy Klasse generiert.

5.6.3. SOAP Implementierungen für PHP

5.6.3.1. NuSOAP

Bei NuSOAP handelt es sich um eine in der Literatur oftmals erwähnte Klassenbibliothek, die es erlaubt, SOAP Web Services unter PHP4 zu erstellen und zu nutzen. Anzumerken ist, dass NuSOAP lediglich bis zum SOAP Standard 1.1 entwickelt wurde. Will man SOAP 1.2 kompatibel sein, so ist man auf PHP5 angewiesen, welches SOAP bereits integriert hat.

NuSOAP Server Implementierung in PHP

Man kann den Server mit und ohne WSDL Erstellung generieren. Zunächst folgt die Implementierung ohne WSDL Erzeugung, die Unterschiede werden im Anschluss besprochen.

```
/* Eingebunden ist NuSOAP sowohl client- als auch serverseitig sehr
schnell, es genügt „require_once ('Verzeichnis/nusoap.php');“ vor die
erste Ausführung einer SOAP Methode zu schreiben */
require_once('lib/nusoap.php');

/* An dieser Stelle werden die Methoden definiert, es müssen dabei keine
Typen angegeben werden */
function add($x, $y){
    return $x + $y;
}

/* als nächstes wird ein SOAP Server Objekt erstellt und dort die Metho-
de add hinzugefügt */
$s = new soap_server;
$s->register('add');

/* Mit HTTP_RAW_POST_DATA werden die Daten an die Service Methode über-
geben, die mit POST an das Web Service geschickt worden sind. Die Inter-
pretierung dieser POST Daten geschieht in der inkludierten Datei
nusoap.php */
$s->service($HTTP_RAW_POST_DATA);
```

Das Web Service ist bereits lauffähig wenn man es über den Web Server anspricht.

PHP NuSOAP Web Service URL:
http://localhost:81/php_projects/nuSOAP/PhpNusoapServer.php

Um mit dem URL Parameter „?wsdl“ eine WSDL Datei erzeugen zu können, muss man dem SOAP Server einige Parameter mehr angeben, die im Folgenden beschrieben sind.

```
/* Die inkludierte Datei und die Funktionsdefinitionen sind identisch
mit denen der WSDL losen Version */
```

```
/* Erneut wird ein SOAP Server Objekt erstellt, allerdings wird die Me-
thode erst zu einem späteren Zeitpunkt beim Service registriert */
$s = new soap_server;

/* Der erste Parameter ist der Web Service Name, der zweite der Name des
Namensraumes */
$s->configureWSDL('PhpNusoapServerWSDL',
'http://localhost:81/php_projects/nuSOAP/');

/* Hier werden die Ein- und Ausgabeparameter definiert */
$input = array('x' => 'xsd:int', 'y' => 'xsd:int');
$output = array('return' => 'xsd:int');

/* An dieser Stelle wird die Methode beim Service registriert, die je-
weiligen Parameter sind kommentiert */
$s->register('add', // Method Name
    $input, // Method Input
    $output, // Method Output
    'http://localhost:81/php_projects/nuSOAP/', // Namespace
    'urn:PhpNusoapServerWSDL#add, // SoapAction
    'rpc', // Style (rpc | document)
    'encoded', // Use (encoded | literal)
    'adds two digits' // Documentation
    );

/* Aufgerufen wird das Service abermals mit dem Aufruf der service Me-
thode */
$s->service($HTTP_RAW_POST_DATA);
```

Nun ist es möglich das Web Service aus dem Browser heraus anzusurfen. Man bekommt, ähnlich wie bei .NET, ein Interface zu sehen, das einem die Methoden sowie einen Link auf die WSDL Datei anzeigt. Die WSDL Datei ist mit „http://<Web Service Url>/WebService.php?wsdl“ aufrufbar.

NuSOAP Client Implementierung in PHP

Wie schon beim Server kann auch der Client mit und ohne WSDL Unterstützung arbeiten. Arbeitet man ohne WSDL, gibt man den Namensraum und die Zielserveradresse manuell an, wie im Folgenden gezeigt wird.

```
/* Auch beim Client reicht eine Inkludierung der nusoap.php aus, um die
SOAP Funktionen nutzen zu können */
require_once('lib/nusoap.php');

/* Die Parameter müssen in ein Array geschachtelt werden */
$parameters = array('x'=>6, 'y'=>3);

/* Zunächst wird ein Client Objekt erzeugt, dem die URL vom Web Service
übergeben wird. Die Erklärung des Zweiers bei der Methode soapclient2
erfolgt am Ende dieses Kapitels */
$soapclient = new
soapclient2('http://localhost:81/php_projects/nuSOAP/PhpNuso
apServer.php');

/* Der Aufruf erfolgt mit der Methode call, die drei Parameter mitgege-
ben bekommt, nämlich den Namen der Methode, die Parameter und den Na-
mensraum */
echo $soapclient->call('add',$parameters,
"http://localhost:81/php_projects/nuSOAP/");
```

Der Client der WSDL Dateien nutzt, unterscheidet sich lediglich in zwei Zeilen, nämlich im Erstellen des soapclient Objekts und im Aufruf des Web Service.

```
/* In diesem Fall gibt man nicht die URL vom Web Service, sondern die
URL der WSDL Datei, an */
$soapclient = new
soapclient2('http://localhost:81/php_projects/nuSOAP/
PhpNusoapServerWSDL.php?wsdl', true);

/* Durch die Informationen in der WSDL Datei kann auf die Angabe des Na-
mensraumes beim Aufruf vom Web Service verzichtet werden */
echo $soapclient->call('add',$parameters);
```

Für den Fall, dass NuSOAP unter PHP5 eingesetzt werden soll, was serverseitig durchaus Sinn macht, da man die WSDL Datei nicht per Hand schreiben muss, ist anzumerken, dass die Klassen „soapclient“ und „soapserver“ in PHP5 belegt sind, wenn SOAP eingeschaltet ist. Man muss diese Klassen daher in der Datei nusoap.php (zum Beispiel in „soapclient2“) zuerst umbenennen. Natürlich muss diese Umbenennung konsequent durch die gesamte nusoap.php Datei hindurch geschehen, da man sonst Fehlermeldungen erhält. Wird unter PHP4 entwickelt, so

kann man sich diese Prozedur ersparen, daher ist es empfehlenswert PHP4 für den Einsatz von NuSOAP zu nutzen, falls dies die gegebene Infrastruktur erlaubt.

5.6.3.2. PHP5 SOAP

Ab PHP5 sind SOAP Funktionen bereits standardmäßig im Basis Paket enthalten. Sowohl Clients als auch Server können mit den beiliegenden Funktionen erstellt werden. Eine detaillierte Auflistung der Funktionen von SOAP unter PHP5 findet man hier: http://at.php.net/manual/de/ref.soap.php. Wichtig ist, dass man SOAP in der php.ini mit folgenden Einstellungen aktiviert:

```
soap.wsdl_cache_enabled = '1';
soap.wsdl_cache_dir = '<basisverzeichnis_xampp>\tmp' (unter
linux '/' statt '\')
soap.wsdl_cache_ttl = '86400'
```

Anschließend kann man die Funktionen ohne Einbindung einer Bibliothek nutzen.

PHP5 Server Implementierung

Bei der PHP5 Implementierung ist derzeit keine automatische WSDL Erstellung implementiert. Das bedeutet, man greift dafür entweder auf andere Produkte wie NuSOAP zurück oder man erstellt die WSDL Datei per Hand.

```
/* An dieser Stelle werden, gleich wie bei NuSOAP, die Methoden defi-
niert. Es müssen dabei keine Typen angegeben werden */
function add($x, $y){
    return $x + $y;
}
```

```
/* Ein SoapServer wird erstellt. Der erste Parameter steht dafür, dass
keine WSDL benutzt wird, der zweite Parameter gibt den Namensraum an,
der durch die fehlende WSDL so definiert werden muss */
$server = new SoapServer(null, array('uri' =>
"http://localhost:81/php_projects/nuSOAP/"));

/* Die add Funktion wird zum Server hinzugefügt */
$server->addFunction("add");

/* Die handle Funktion ist das Gegenstück zur service Funktion der vor-
herigen Implementierung und wird ausgeführt wenn das Web Service aufge-
rufen wird */
$server->handle();
```

Will man einen Server unter Zuhilfenahme von Daten aus einer WSDL Datei erstellen, so ändert man den SoapServer Aufruf wie folgt:

```
$server = new SoapServer('PhpSoapServerWSDL.wsdl');
```

In diesem Fall empfiehlt es sich das Cachen der WSDL Datei mit dem Befehl `ini_set("soap.wsdl_cache_enabled", "0");` zu deaktivieren.

PHP5 Client Implementierung

Auch beim PHP5 Client gibt es die Möglichkeit das XML einer WSDL Datei auszulesen oder das Web Service direkt anzusprechen.

```
/* Gibt man keine WSDL Datei an, so muss man die Parameter Web Service
Adresse, Namensraum, Dokumentstil und Dokumentverarbeitung bei der Er-
stellung des SoapClient Objektes angeben. */
$client = new SoapClient(NULL,
array(
    "location" =>
"http://localhost:81/php_projects/nuSOAP/PhpSoapServer.php",
    "uri"      => "urn:HelloServiceSoap",
    "style"    => SOAP_RPC,
    "use"      => SOAP_ENCODED
));

/* Der Aufruf erfolgt mit der __call() Methode, wobei der erste Parame-
ter der Methodenname und der zweite ein Array ist, in dem die Übergabe-
parameter verpackt sind */
echo $client->__call("add", array(5, 9));
```

Benutzt man eine WSDL Datei, so ändert sich die Client Erstellung wie folgt.

```
/* Dem Soap Client wird lediglich das WSDL Dokument bei der Erstellung
übergeben */
$soapclient = new
SoapClient("http://localhost:8180/axis/services/SoapServer?w
sdl");

/* Die Parameter werden als SoapParam Objekte übergeben, wobei anzumer-
ken ist, dass der erste Parameter bei der SoapParam Erstellung der Wert
und der zweite Parameter die Bezeichnung darstellt */
$parameters = array(new SoapParam($x,"a"), new
SoapParam($y,"b"));

/* Der Aufruf des Web Service erfolgt über die __call Funktion */
echo $soapclient->__call("add", $parameters);
```

Verwendet man eine WSDL Datei, so kann man mit der Methode `$soapclient->__getFunctions()` die Methoden der WSDL Datei ausgeben lassen.

6. Auswertung und Interpretation

6.1. Evaluierung der Interoperabilität der XML-RPC Implementierungen

Die fünf näher betrachteten Implementierungen von XML-RPC Web Services, nämlich Apache XML-RPC (Java), Rox (Java), XmlrpcCS (.NET), PhpXmlrpc20 (PHP) und IXR (PHP), wurden sowohl client- als auch serverseitig gegeneinander getestet und sind in folgender Matrix dargestellt.

XML-RPC		Server: Apache	Server: Rox	Server: XmlrpcCS	Server: PhpXmlrpc20	Server: IXR
Clients	Apache	+	+	+	!-	+
					Fb.: 5	
	Rox	+	+	!-	!-	+
				Fb.: 3	Fb.: 5	
	XmlrpcCS	+	!-	+	!-	+
			Fb.: 2		Fb.: 5	
	PhpXmlrpc20	+	!-	!+	+	+
			Fb.: 1	Fb.: 4		
	IXR	+	!-	!+	!-	+
			Fb.: 1	Fb.: 4	Fb.: 5	

Legende			
	+		Funktioniert ohne Probleme
	!+		Funktioniert zwar, jedoch nicht ohne Probleme
	!-		Funktioniert nicht
	Fb.: X		Fehlerbeschreibung: X

Abbildung 09 – Matrix der Interoperabilität von XML-RPC Implementierungen

Bei den Web Service Integrationen heben sich die Apache und die IXR Implementierung serverseitig deutlich von allen anderen ab. Wie man in der XML-RPC Matrix erkennen kann, gibt es bei diesen beiden Web Services keinen Client, der die XML Daten der Server falsch interpretiert.

Clientseitig ist ebenfalls der Apache der Testsieger, da er nur mit dem PhpXmlrpc20 Web Service Probleme aufweist, auf die im Folgenden näher eingegangen wird.

XML-RPC Matrix – Fehlerbeschreibung: 1

Beim Aufruf des Rox Servers mit den PHP Clients wird korrektes XML als Antwort des Aufrufs der Math.add Methode empfangen, jedoch beendet der PHP Client die Anfrage nach 60 Sekunden mit der Fehlermeldung „Maximum execution time of 60 seconds exceeded", noch bevor die empfangenen Daten interpretiert werden können.

```
01  POST /web/xmlrpc HTTP/1.0
02  Host: localhost
03  Content-Type: text/xml
04  User-Agent: The Incutio XML-RPC PHP Library
05  Content-length: 209
```

Code 12 – HTTP Header der IXR Client Anfrage:

```
01  POST /xmlrpc/ HTTP/1.0
02  User-Agent: XML-RPC for PHP 2.0
03  Host: localhost
04  Accept-Encoding: gzip, deflate
05  Accept-Charset: UTF-8,ISO-8859-1,US-ASCII
06  Content-Type: text/xml
07  Content-Length: 191
```

Code 13 – HTTP Header der PhpXmlrpc20 Client Anfrage:

An diesen Headern kann man in Zeile 1 erkennen, dass beide Implementierungen ihre Anfragen via HTTP Version 1.0 senden. Vergleicht man in weiterer Folge die Header Daten der Antwort vom Apache Service mit denen vom Rox Service, erhält man Aufschluss über die mögliche Ursache der Zeitüberschreitungen der PHP Clients.

```
01  HTTP/1.1 200 OK
02  Server: Apache-Coyote/1.1
03  Content-Type: text/xml
04  Content-Length: 129
05  Date: Tue, 24 Jul 2006 08:24:15 GMT
06  Connection: close
```

Code 14 – HTTP Header der Antwort vom Apache Server

```
01  HTTP/1.1 200 OK
02  Host: localhost
03  Server: RoX/0.7 Windows XP 5.1 (x86)
04  Content-Type: text/xml
05  Content-Length: 131
06  Date: Di, 24 Jul 2006 10:14:19 +0200
```

Code 15 – HTTP Header der Antwort vom Rox Server

Der Antwort HTTP Header von Rox enthält keinen Parameter „Connection:close" der die Verbindung beendet, während Apache die Verbindung in Zeile 6 schließt. Bei HTTP 1.1 ist das default Verhalten für die Verbindung „keep-alive". Da beide PHP Clients ihre Anfragen mit HTTP 1.0 senden und bei beiden das gleiche Problem auftritt, kann man davon ausgehen, dass die HTTP Version 1.1 nicht vollständig unterstützt wird, d.h. die Verbindungsart „keep-alive" bei der Rox Antwort das Problem der Zeitüberschreitung darstellt. Die XML Daten der ersten Anfrage wurden zwar vom Server korrekt gesendet, die Clients warten jedoch vergeblich auf die Beendigung der Verbindung bevor sie die Daten verarbeiten.

XML-RPC Matrix – Fehlerbeschreibung: 2

Beim Versuch, mit dem XmlrpcCS Client von .NET auf das Rox Web Service zuzugreifen, tritt ein Absturz des Programms auf.

```
01  POST /xmlrpc HTTP/1.1
02  Content-Type: test/xml
03  Host: localhost:5555
04  Content-Length: 197
05  Expect: 100-continue
06  Connection: Keep-Alive
```

Code 16 – HTTP Header der XmlrpcCS Client Anfrage

```
01  HTTP/1.1 500 Internal Server Error
02  [...]
03  Content-Length: 0
```

Code 17 – HTTP Header der Antwort vom Rox Server

Die Anfrage des .NET Clients kommt mit dem sehr untypischen „Content-Type" Parameter „test/xml" beim Rox Server an, mit dem dieser Server nicht arbeiten kann und deshalb mit einer „internen Server Error" Fehlermeldung antwortet. Der korrekte „Content-Type" Parameter lautet „text/xml".

XML-RPC Matrix – Fehlerbeschreibung: 3

Beim Aufruf des XmlrpcCS Web Service mit dem Rox Client wird die Methode Math.add auf dem Server ausgeführt und auch in korrektem XML zurück an den Client gesendet. Der Rox Client erzeugt jedoch bei Empfang der Antwort eine „RPC call failed" Ausnahme, mit der Begründung, dass er auf eine Antwort vom Server wartet.

```
01  HTTP/1.1 200 OK
02  Connection: close
03  Server: XmlRpcServer
04  Content-Type: text/xml
```

Code 18 – HTTP Header der Antwort vom XmlrpcCS Server

Grund dafür könnte sein, dass der Server kein Header Attribut „Content-Length" mitsendet, was aber wie in Punkt 1.4.1.2 beschrieben, der Fall sein muss damit ein HTTP Response korrekt verarbeitet werden kann.

XML-RPC Matrix – Fehlerbeschreibung: 4

Bei der Verbindung zum XmlrpcCS Server haben beide PHP Clients ein Problem bei der Verarbeitung von <double> Parametern. Die ersten drei Anfragen werden von beiden Clients korrekt verarbeitet. Bei der Verarbeitung des double Parameters gibt es jedoch das Problem, dass die Werte von den Implementierungen nicht korrekt in das Empfangsarray umgewandelt werden. Vergleicht man die XML Struktur einer Antwort vom IXR Server mit der des XmlrpcCS Servers, so findet man lediglich einen Unterschied in der XML Definition, welche bei der .NET Web Service Antwort „encoding=utf-8" enthält, nicht aber bei der IXR Antwort.

```
01  <?xml version="1.0" encoding="utf-8"?>
02  <methodResponse><params>
03  <param><value><double>0,6</double></value></param>
04  </params></methodResponse>
```

Code 19 – HTTP Body der Antwort vom XmlrpcCS Server

```
01  <?xml version="1.0"?>
02  <methodResponse><params>
03  <param><value><double>0.6</double></value></param>
04  </params></methodResponse>
```

Code 20 – HTTP Body der Antwort vom IXR Server

Da der <double> Parameter bei der IXR Server Antwort von den Clients korrekt ausgegeben wird, kann man annehmen, dass das XML je nach „encoding" Parameter in den PHP Client Implementierungen unterschiedlich abgearbeitet wird und beim Enkodieren nach utf-8 ein Programmierfehler in den Implementierungen auftritt, der den Parameter anders in das „Result" Array verpackt.

XML-RPC Matrix – Fehlerbeschreibung: 5

Betrachtet man den PhpXmlrpc20 Server, so sieht man, dass dieser mit keinem Client außer dem eigenen arbeiten kann. Der Grund dafür liegt im XML der Antwort.

```
01  HTTP/1.1 200 OK
02  Date: Tue, 01 Aug 2006 12:26:08 GMT
03  Server: Apache/2.2.0 (Win32) DAV/2 mod_ssl/2.2.0
    OpenSSL/0.9.8a mod_autoindex_color PHP/5.1.1
04  X-Powered-By: PHP/5.1.1
05  Vary: Accept-Charset
06  Content-Length: 242
07  Connection: close
08  Content-Type: text/xml
```

Code 21 – HTTP Header der Antwort vom PhpXmlrpc20 Server

```
01  <?xml version="1.0"?>
02  <methodResponse><params>
03  <param><value><int>13</int></value></param>
04  </params></methodResponse><?xml version="1.0"?>
05  <methodResponse><params>
06  <param><value><int>13</int></value></param>
07  </params></methodResponse>
```

Code 22 – HTTP Body der Antwort vom PhpXmlrpc20 Server

Das XML Dokument wird zweimal im Body der HTTP Antwort mitgeschickt, was zu einem Syntaxfehler bei allen fremden Clients führt. Der Grund warum der PhpXmlrpc20 Client die Daten darstellen kann, liegt

darin, dass die Daten von der Implementierung anders verarbeitet und verschickt werden.

```
01  HTTP/1.1 200 OK
02  Date: Tue, 01 Aug 2006 12:24:47 GMT
03  Server: Apache/2.2.0 (Win32) DAV/2 mod_ssl/2.2.0
    OpenSSL/0.9.8a mod_autoindex_color PHP/5.1.1
04  X-Powered-By: PHP/5.1.1
05  Vary: Accept-Encoding
06  Content-Encoding: gzip
07  Content-Length: 232
08  Connection: close
09  Content-Type: text/xml
```

Code 23 – HTTP Header der Antwort vom PhpXmlrpc20 Server bei einem PhpXmlrpc20 Client Aufruf

```
01  ‹□
```

Code 24 – HTTP Body der Antwort vom PhpXmlrpc20 Server bei einem PhpXmlrpc20 Client Aufruf

Der Server erkennt, dass die Anfrage von einem PhpXmlrpc20 Client verschickt wurde und wandelt das XML mit einem anderen Algorithmus um, der die Nachricht gleich in „gzip" Format enkodiert. Dieses Format lässt sich im Tunnel Programm nur durch „‹□" darstellen.

6.2. Evaluierung der Interoperabilität der SOAP Implementierungen

Getestet wurden die SOAP Implementierungen Axis (Java), Apache (Java), die Microsoft.NET SOAP Integration (.NET), NuSOAP (PHP) und PhpXmlrpc20 (PHP).

Bei den SOAP Web Services muss man zwischen WSDL unterstützten Web Services (im Folgenden kurz „WSDL Web Services" genannt) und Services ohne WSDL Unterstützung unterscheiden. Man kann zwar, wie man in der SOAP Matrix sieht, WSDL Web Services mit nicht WSDL Clients ansprechen, jedoch ist es nicht möglich, Web Services die ohne WSDL arbeiten, mit Clients die auf WSDL Unterstützung angewiesen sind, anzusprechen.

Die Axis Implementierung bietet drei Möglichkeiten an, Web Services mit WSDL Unterstützung online zu stellen. Axis WSDL mit RPC/Encoded formatierten Nachrichten, Axis WSDL mit Document/Literal formatierten Nachrichten und Axis WSDL .jws, welches auf sehr einfache Weise erstellt werden kann. Die PHP Clients NuSOAP und PHP5 bieten Clients und Server sowohl mit WSDL Unterstützung, als auch ohne an. Die Implementierung von Apache bietet hingegen keine WSDL Unterstützung an.

SOAP		Server								
		Axis WSDL Encoded	Axis WSDL Literal	Axis WSDL .jws	.NET WSDL	NuSOAP WSDL	PHP5 WSDL	Apache	NuSOAP	PHP5
Clients	Axis WSDL	+	+	+	+	+	+	–	–	–
	.NET WSDL	!+	+	!+	+	!+	!+	–	–	–
		Fb.: 3		Fb.: 3		Fb.: 3	Fb.: 3			
	NuSOAP WSDL	+	!-	+	!-	+	+	–	–	–
			Fb.: 6		Fb.: 6					
	PHP5 WSDL	+	!+	+	!+	+	+	–	–	–
			Fb.: 5		Fb.: 5					
	Axis	+	+	+	!-	+	+	!-	+	+
					Fb.: 4			Fb.: 2		
	Apache	+	!-	+	!-	+	+	+	+	+
			Fb.: 1		Fb.: 4					
	NuSOAP	+	+	+	!-	+	+	+	+	+
					Fb.: 4					
	PHP5	+	+	+	!-	+	+	+	+	+
					Fb.: 4					

Legende:			
	+		Funktioniert ohne Probleme
	!+		Funktioniert zwar, jedoch nicht ohne Probleme
	!-		Funktioniert nicht
	–		Kann technisch nicht funktionieren
	Fb.: X		Fehlerbeschreibung: X

Abbildung 10 – Matrix der Interoperabilität von XML-RPC Implementierungen

SOAP Matrix – Fehlerbeschreibung: 1

Der Java Apache Client ohne WSDL Unterstützung hat sowohl beim Web Service Aufruf vom Axis WSDL Literal Web Service, als auch beim Aufruf des .NET Servers Probleme. Die Server enkodieren ihre Nach-

richten in Dokument/Literal Format, was darauf schließen lässt, dass Apache aus diesem Grund die Daten nicht verarbeiten kann und deswegen mit der Fehlermeldung „no deserializer found" reagiert.

SOAP Matrix – Fehlerbeschreibung: 2

Ein sehr interessanter Fehler ist, dass der Axis Client ohne WSDL Unterstützung beim Apache Web Service ebenfalls Probleme beim „Deserializen" aufzeigt. Diese Fehlermeldung tritt jedoch bei keinem anderen RPC/Encoded Web Server auf, der mit dem Axis Client aufgerufen wird. Der Grund für dieses Problem ist, dass „<multiRef>" Attribute erst ab SOAP Version 1.2 implementiert werden müssen, Apache jedoch nur SOAP 1.1 unterstützt.

```
01  <?xml version="1.0" encoding="UTF-8"?>
02  <soapenv:Envelope
    xmlns:soapenv="http://schemas.xmlsoap.org/soap/envelope/
    " xmlns:xsd="http://www.w3.org/2001/XMLSchema"
    xmlns:xsi="http://www.w3.org/2001/XMLSchema-instance">
03  <soapenv:Body>
04  <ns1:add
    soapenv:encodingStyle="http://schemas.xmlsoap.org/soap/e
    ncoding/" xmlns:ns1="urn:soapserver">
05      <ns1:arg0 href="#id0"/>
06      <ns1:arg1 href="#id1"/>
07  </ns1:add>
08  <multiRef id="id1" soapenc:root="0"
    soapenv:encodingStyle="http://schemas.xmlsoap.org/soap/e
    ncoding/" xsi:type="soapenc:int"
    xmlns:soapenc="http://schemas.xmlsoap.org/soap/encoding/
    ">5</multiRef>
09  <multiRef id="id0" soapenc:root="0"
    soapenv:encodingStyle="http://schemas.xmlsoap.org/soap/e
    ncoding/" xsi:type="soapenc:int"
    xmlns:soapenc="http://schemas.xmlsoap.org/soap/encoding/
    ">8</multiRef>
10  </soapenv:Body>
11  </soapenv:Envelope>
```

Code 25 – SOAP Body der Anfrage vom Axis Client zum Apache Server

Die Dokumentation spezifiziert eine Eigenschaftsvariable namens „ENCODINGSTYLE_URI_PROPERTY“, die definiert welche SOAP Version zum Enkodieren der Nachrichten verwendet werden soll.

SOAP Matrix – Fehlerbeschreibung: 3

Das Microsoft Framework .NET erzeugt ausschließlich Document/Literal formatierte WSDL Dateien für Web Services, kann aber clientseitig auch RPC/Encoded Varianten interpretieren. Will man mit der Datei wsdl.exe einen Proxy für seinen Client erzeugen, so erhält man eine Warnung bei allen nicht „WS-I Basic Profile Version 1.0“ konformen WSDL Dateien, wozu das Format RPC/Encoded zählt. Die Proxies für den Web Service Zugriff werden trotz dieser Warnmeldung erstellt und sind funktionstüchtig.

SOAP Matrix – Fehlerbeschreibung: 4

Das .NET Web Service verlangt einen gültigen HTTP Parameter namens „SOAPAction“, der aus der WSDL Datei ausgelesen werden kann. Verwendet man jedoch Clients ohne WSDL Unterstützung, muss man diesen Parameter, sofern dieser in der Implementierung geändert werden kann, manuell setzen.

Beim PHP5 Client kann man SOAPAction laut Dokumentation als Parameter beim Erstellen des Clientobjekts angeben, allerdings wird dieser beim Umwandeln in XML ignoriert und stattdessen der Methodenname mit „#Methodenname“ an die angegebene URI Adresse hinzugefügt. Der .NET Server verlangt die SOAPAction jedoch mit Postfix „/Methodenname“.

Bei den Clients Apache, Axis und NuSOAP funktionieren zwar die SOAPAction Zuweisung mit „/Methodenname“, dafür können die Parameter vom .NET Service nicht richtig interpretiert werden, womit immer 0 zurückgesendet wird.

SOAP Matrix – Fehlerbeschreibung: 5

Beide PHP WSDL Web Services enkodieren ihre Nachrichten standardmäßig nach dem RPC/Encoded SOAP Format. Die RPC/Encoded Formatierung hat zur Folge, dass man bei der PHP5 Implementierung Document/Literal Web Service anders ansprechen muss als RPC/Encoded Web Services. Die Parameter werden bei einem RPC/Encoded Aufruf mit „`$parameters = array(new SoapParam($x,"a"), new SoapParam($y,"b"))`“ gesetzt und die Methode auf dem Zielserver mit „`$soapclient->__call("add", $parameters)`“ aufgerufen. Bei einem Aufruf eines Document/Literal Web Service müssen die Parameter mit „`$parameters = array('a' => $x, 'b' => $y);`“ gesetzt und die Methoden mit „`$soapclient->add($parameters)->addResult`“ definiert werden, wobei der Name des Rückgabeparameters „addResult“ von der Methode und von der Implementierung abhängt.

SOAP Matrix – Fehlerbeschreibung: 6

Ein Problem der NuSOAP Client Variante mit WSDL Unterstützung ist, dass die Parameter bei Anfragen zu Document/Literal Web Services anders verpackt werden müssen als bei Anfragen zu einem RPC/Encoded Web Services. Document/Literal nutzende Web Services verlangen die Daten im „Struct“ (assoziativen Array) Format, während Web Services die RPC/Encoded benutzen, ein nicht assoziatives Array als Parameter benötigen. Für den Aufruf der PHP Web Services macht die Art des Parameterarrays keinen Unterschied.

Auch mit der Parameterstrukturierung als Struct lassen sich das .NET und das Axis WSDL Literal Web Service nicht korrekt mit dem NuSOAP Client aufrufen, denn obwohl die NuSOAP Dokumentation bestätigt, dass die Parameter als assoziatives Array an die „call" Funktion übergeben werden können, sind keine Parameter im SOAP Body bei der Client Anfrage vorhanden, was auf einen Fehler in der Client Implementierung hindeutet.

```
01  <?xml version="1.0" encoding="ISO-8859-1"?>
02  <SOAP-ENV:Envelope xmlns:SOAP-
    ENV="http://schemas.xmlsoap.org/soap/envelope/"
    xmlns:xsd="http://www.w3.org/2001/XMLSchema"
    xmlns:xsi="http://www.w3.org/2001/XMLSchema-instance"
    xmlns:SOAP-
    ENC="http://schemas.xmlsoap.org/soap/encoding/"
    xmlns:ns6642="urn:onjavaserver">
03      <SOAP-ENV:Body>
04          <add xmlns="http://DefaultNamespace"/>
05      </SOAP-ENV:Body>
06  </SOAP-ENV:Envelope>
```

Code 26 – SOAP Body der Anfrage vom NuSOAP Client zum Axis WSDL Literal Server

6.3. Erstellung von Hypothesen

Im Laufe der Durchführung war auffallend, dass Implementierungen mit einer großen Community leichter integriert werden können als Implementierungen, die eine weniger große Anhängerschaft haben. Das ist wahrscheinlich darauf zurückzuführen, dass es umfangreichere Dokumentationen für bekanntere Implementierungen gibt. Umfangreichere Dokumentationen lassen wiederum ein schnelles Beheben von Fehlverhalten der Applikation zu.

Dies führt zu folgenden Hypothesen, die anderen Arbeiten zu Grunde liegen könnten:

Je besser eine Implementierung dokumentiert ist, umso wahrscheinlicher ist es, dass sie sich ohne viel Aufwand integrieren lässt.

Die Verbreitung einer Implementierung sagt nichts über die Wahrscheinlichkeit aus, mit der diese interoperabel mit anderen Implementierungen ist.

6.4. Beantwortung der Forschungsfragen

6.4.1. Ad. 3.1 Forschungsfrage nach der Integration

Wie unterscheiden sich verschiedene SOAP/XML-RPC Implementierungen unterschiedlicher Programmiersprachwelten von einander im Hinblick auf die Erstellung von Web Services?

6.4.1.1. Integrationsaufwand

Die drei unterschiedlichen Plattformen haben allesamt eigene, aber ähnliche Wege gefunden, Web Services zu integrieren. Während Java Bibliotheken die Implementierungen in sehr modulare Klassen aufteilen, sieht man bei PHP meist ein oder zwei Sammelbibliotheken, die sich um alles kümmern. Im Prinzip läuft es jedoch bei allen Implementierungen ähnlich ab. Man importiert eine bestehende Bibliothek und kann aus den Klassen in weiterer Folge Objekte erstellen, die sich um das Verpacken der Nachricht kümmern. Komplizierter wird es dann, wenn, wie bei der Java Apache XML-RPC Bibliothek, Abhängigkeiten auf Pakete, die nicht gängig sind, auftreten.

Unterschiede in der Web Service Erstellung gibt es, sowohl in XML-RPC als auch in SOAP spezifizierten Paketen, vor allem im Server Bereich. Betrachtet man die PHP Implementierung, so ist diese lediglich ein File, das beim Aufruf der Server Datei inkludiert wird und die Übersetzung in das XML Format übernimmt. Anders als bei den Java Implementierungen, wo im Web Server klar definiert sein muss, unter welcher URI sich das Servlet oder der Deployment Descriptor befindet, ruft man hier direkt die Klassen auf, die die Methoden definieren.

6.4.1.2. Toolunterstützung

Toolunterstützungen gibt es vor allem im SOAP Bereich, da zum Beispiel das Erstellen von WSDL Dateien per Hand sehr aufwendig ist.

Das von Java Axis angebotene Java2WSDL ist ein sehr nützliches Tool bei der Erstellung von SOAP Web Services. Es interpretiert den Java Code einer Klasse und schreibt aufgrund der Methoden das passende WSDL File.

Das vom .NET Framework angebotene Pendant dazu geht sogar noch einen Schritt weiter. Die verwendete wsdl.exe erstellt eine WSDL Datei aus dem Code der Web Service Klasse, mit der Technologie namens „Reflektion", die das Web Service auch gleich zur Laufzeit in den Web Server einbindet und außerdem automatisch generierte Testclients für jede Web Service Methode anbietet.

Die PHP NuSOAP Implementierung bietet ebenfalls die Möglichkeit WSDL Dateien automatisch aus der Web Service Klasse heraus erstellen zu können. Dieses Feature ist direkt in der NuSOAP Bibliothek Datei eingebunden. Bei der PHP5 Implementierung von SOAP musste die WSDL Datei hingegen per Hand erstellt werden. Die Implementierung

soll jedoch, laut Dokumentation, in naher Zukunft um ein Tool das WSDL schreiben kann, erweitert werden.

Clientseitig bieten das .NET Framework und die Axis Implementierung von Java eine automatische Proxy Erstellung, anhand der Vorlage einer WSDL Datei, an. Dadurch werden lokale Objekte erzeugt, die das Web Service mit all seinen Methoden repräsentieren. Die Kommunikation mit dem Web Service wird vom Proxy übernommen.

6.4.1.3. Dokumentation

Bei der Dokumentation gibt es sehr große Unterschiede zwischen den einzelnen Implementierungen, dennoch lässt sich sagen, dass im Rahmen dieser Arbeit im SOAP Bereich mehr Unterstützung beim Integrieren der Implementierungen gefunden wurde als im XML-RPC Bereich. XmlrpcCS (XML-RPC/.NET) hat zum Beispiel keine funktionierende online Dokumentation mehr, was wohl auf das Alter der Implementierung zurückzuführen ist. Neuere Implementierungen wie Axis von Java oder die PHP5 SOAP Implementierung bieten hingegen einen sehr großen Dokumentationsumfang an.

In welchem Rahmen sich der Umfang der Gemeinschaft um eine Implementierung und die Fülle der Dokumentation auf die Implementierungsqualität auswirkt, wäre ein interessanter Ansatz für eine Folgediplomarbeit.

6.4.2. Ad. 3.2 Forschungsfrage nach der Interoperabilität

Welche Diskrepanzen treten auf, wenn man mit unterschiedlichen Programmiersprachen eine einheitliche, interoperable Funktion durch Web Services erreichen will?

Dazu muss man zwischen der XML-RPC und der SOAP Spezifikation unterscheiden.

6.4.2.1. Probleme mit XML-RPC

Bei der Auswertung der XML-RPC Web Services scheint deutlich auf, dass die Interoperabilität selten an der Verpackung der Daten in die XML Struktur scheitert, sondern die von den Implementierungen erzeugten Header Daten in der HTTP Übertragung fehlen oder falsch gesetzt sind. Sofern die Implementierung das manuelle Setzen der Header Daten unterstützt, sollte man dies auf jeden Fall nutzen, um solche Interoperabilitäten im Vorhinein ausschließen zu können.

6.4.2.2. Probleme mit SOAP

Ein Problem mit der SOAP Kommunikation wird vom HTTP Header Parameter „SOAPAction" verursacht, der dazu eingesetzt werden kann SOAP Nachrichten als solche zu identifizieren und durch Firewalls zu schleusen. Der .NET Server verlangt bei einem Web Service Aufruf einen gültigen SOAPAction Parameter, der in der WSDL ausgelesen werden kann. Das Problem tritt dann auf, wenn man das Web Service mit nicht WSDL unterstützenden Clients konsumieren möchte, da der SOAPAction Parameter nicht in allen der getesteten Clients manuell gesetzt werden kann. Entscheidet man sich für eine WSDL lose Client

Implementierung sollte man daher unbedingt darauf achten, dass diese die Möglichkeit der SOAPAction Definition anbietet.

Was die unterschiedlichen Nachrichtenformate RPC/Encoded und Document/Literal betrifft, so haben die Tests ergeben, dass die meisten Web Service Implementierungen beide Formate verarbeiten können. Nur der NuSOAP WSDL und der Apache Client, die beide schon etwas älter sind, haben in diesem Bereich starke Mängel aufgezeigt.

6.4.2.3. Die Relevanz der WSDL Datei

Sieht man von der fehlerhaften NuSOAP Client Implementierung ab, so konnten alle WSDL nutzenden Web Services von allen WSDL nutzenden Clients aufgerufen werden. Auch die nicht WSDL nutzenden Clients konnten die WSDL Web Services, sofern diese RPC/Encoded waren, problemlos aufrufen. Die Resultate sprechen stark für die Nutzung von WSDL Dateien wenn man mit SOAP Web Services arbeitet.

6.4.3. Ad. 3.3 Forschungsfrage nach der Spezifikation

Warum wird heutzutage immer noch mit XML-RPC gearbeitet, obwohl SOAP eine daraus resultierende Weiterentwicklung darstellt?

Obwohl XML-RPC bereits eine relativ alte Spezifikation darstellt, erfreut sie sich einer großen Fan-Gemeinschaft. Das liegt daran, dass das XML-RPC XML Format mit seinen wenigen Datentypen sehr simpel gehalten ist und der Datenaustausch auf der anderen Seite genau dadurch sehr schnell abläuft. Die Kehrseite ist offensichtlich. Komplexere Daten können damit auf einfache Weise nicht übertragen werden. SOAP bietet da weit mehr Möglichkeiten, wie zum Beispiel das Senden von ganzen Dokumenten. Auch unterstützt SOAP im Gegensatz zu XML-RPC andere

Protokolle neben HTTP, wie zum Beispiel POP3 und SMTP. Ein Nachteil bei XML-RPC ist das Fehlen der WSDL Datei. Ist eine WSDL Datei für SOAP vorhanden, so kann man diese automatisch auslesen und seinen Client mit den Meta Daten versorgen, die er braucht um das Web Service zu konsumieren. Bei XML-RPC muss diese Information anderweitig zum Konsumierenden gelangen, wie zum Beispiel über eine gute Dokumentation.

(vgl. http://www.phpmagazin.de/itr/online_artikel/psecom,id,649,nodeid,62.html, 01.08.2006)

7. Zusammenfassung

Ob man XML-RPC oder SOAP für das eigene Web Service verwenden soll, hängt stark davon ab, was für Anforderungen man an das Web Service stellt. Sollen komplexe Dokumente ausgetauscht werden oder einfache Daten? Soll die Kommunikation durch ein WSDL File definiert werden oder soll der Konsumierende die Zugänge aus Sicherheitsgründen erst durch eine Dokumentation erhalten? Diese Fragen spielen bei der Entscheidung eine wichtige Rolle. Auch die Frage welche Spezifikation häufiger im Kundenkreis verwendet wird, sollte bereits bei der Auswahl der Spezifikation für ein Web Service bedacht werden. Möchte man ein Web Service nutzen, muss man sich nach der Spezifikation richten, die dieses vorgibt.

Welche Implementierung man nutzen sollte, hängt stark von den Gegebenheiten ab, die man im Arbeitsbereich zur Verfügung hat. Wenn man seine Entwicklungsumgebung dahingehend beeinflussen kann, empfiehlt es sich aber für XML-RPC Web Services die Apache Implementierung für Java oder die IXR Implementierung für PHP zu nutzen, da alle getesteten Clients mit diesen beiden Web Service Implementierungen problemlos Daten austauschen konnten.

Im SOAP Bereich konnten die Axis Web Services von Java sowie beide der getesteten PHP Web Service Implementierungen durch Interoperabilität überzeugen. Es empfiehlt sich bei SOAP Web Services immer eine WSDL Datei anzubieten, da man den Konsumierenden damit eine breitere Auswahl an Clients ermöglicht, mit denen das Web Service genutzt werden kann.

8. Glossar

Bibliothek: Bibliotheken (engl. Libraries) sind eine Ansammlung von Klassen, die man in Projekte importieren und wie eigene Klassen benutzen kann.

Body: Im Body einer HTTP Nachricht wird die eigentliche Information gesendet. Dabei kann es sich zum Beispiel um eine HTML oder XML Datei handeln.

Client: Als Client wird ein Endgerät bezeichnet, das Daten auf einem Server aufruft und dem Endkunden präsentiert.

Compiler: Compiler ist eigentlich ein englisches Wort, dass jedoch in der Technik-Welt oft eingesetzt wird. Ein Compiler wandelt Quellcode Dateien in binäre Zeichenfolgen um, die vom Prozessor lesbar sind. Unterschiedliche Prozessoren und Frameworks haben unterschiedliche Compiler.
(vgl. Albahari, Drayton, Neward, 2003, [eBook] Kapitel 1.2.2)

Deployment Descriptor: Ein Deployment Descriptor ist eine XML Datei, die Informationen über ein Service und den Pfad zur Java Klasse, das dieses Service implementiert, enthält.
(vgl. Englander, 2002, S53)

Framework: Unter Framework kann man im Software Bereich einen Rahmen verstehen, in den Applikationen eingebunden sind. Es ist als Grundbaustein eines Softwaresystems anzusehen, auf dem Anwendungen eine gemeinsame Basis finden.

Header: Im Header einer HTTP Nachricht werden Informationen gespeichert, die den Empfänger wissen lassen, welchen Inhalt er im Body zu erwarten hat. Es wird also zunächst immer der Header einer Datei betrachtet und erst im Anschluss daran der Body.

Host: Unter Host kann man im Zusammenhang dieser Diplomarbeit einen Computer verstehen, der Dienste in einem Netzwerk anbietet.
(vgl. http://de.wikipedia.org/wiki/Host_%28Informationstechnik%29, 09.05.2006)

HTTP: Das HyperText Transfer Protocol ist das Standard Protokoll des World Wide Web. HTTP ist essenziell für Web Services, da die meisten Firewalls von Firmen die Kommunikation über HTTP Port 80 erlauben.
(vgl. Duthie, MacDonald, 2003, [eBook] Kapitel 4.1)

Mapping: Unter „Mapping" versteht man den Mechanismus der benutzt wird um SOAP bzw. XML-RPC Typen in Typen der jeweiligen Programmiersprache umzuwandeln. Zum Beispiel wird in Java bei der Rox Implementierung ein XML-RPC Typ „<struct>" in ein „Map" Objekt umgewandelt. Mapping umfasst natürlich auch die umgekehrte Transformation.

Metadaten: Metadaten sind Daten, die Aussagen über eine Nachricht geben. Zum Beispiel kann der Nachrichtentyp, die Länge oder die Kodierung der Nachricht als Metadaten abgespeichert werden.

Middleware: Unter Middleware versteht man die Verbindung zweier Software Architekturen. Eine Möglichkeit, Daten zwischen den Software Produkten und der Middleware auszutauschen, sind Web Services.
(vgl. St. Laurent, Johnston, Dumbill, 2001, S82)

Namensraum: Namensräume (engl. Namespaces) dienen dazu, Ordnung in Klassenansammlungen zu bringen und zusammengehörige Klassenbibliotheken hierarchisch zu sortieren.

Plattform: Unter Plattform wird in dieser Arbeit die Programmierumgebung verstanden. Beispiele für solche Umgebungen sind Java, das .NET Framework und PHP.

Reflektion: Die vom .NET Framework benutzte Technologie Reflektion kann existierende Assemblies und Typen untersuchen und im Echtzeitbetrieb ins Framework einbinden.
(vgl. Albahari, Drayton, Neward, 2003, [eBook] Kapitel 13)

Request: Ein „Request" ist eine Anfrage die ein Client an einen Server stellt.

Response: Ein „Response" ist eine Antwort die ein Server einem Client gibt.

Schnittstelle: „Eine Schnittstelle (englisch interface) ist ein Teil eines Systems, das dem Austausch von Informationen als Spannungen, Energie oder Materie analog oder digital mit anderen Systemen dient." … „Eine Schnittstelle wird durch eine Menge von Regeln beschrieben, der Schnittstellenbeschreibung. Neben der Beschreibung, welche Funktionen vorhanden sind und wie sie benutzt werden, gehört zu der Schnittstellenbeschreibung auch ein so genannter Kontrakt, der die Semantik der einzelnen Funktionen beschreibt."
(http://de.wikipedia.org/wiki/Schnittstelle, 09.05.2006)

Serialisieren: Serialisation (engl. Serialization) transformiert ein Objekt in einen Bytestrom, der über ein Netzwerk gesendet werden kann. Die Umkehrform nennt man „Deserialisation".
(vgl. Albahari, Drayton, Neward, 2003, [eBook] Kapitel 38)

Tag: Als Tags bezeichnet man die Strukturierungselemente in XML bzw. HTML Dateien. Ein Tag hat stets einen Anfang <Tag> und ein Ende </Tag>. Ist ein Tag leer, so kann er auch im Anfangstag beendet werden, <Tag/>.

TCP (Transmission Control Protocol): Es handelt sich dabei um eine „Vereinbarung" zwischen Computern, wie Daten untereinander ausgetauscht werden sollen. Da TCP auf das Internet Protokoll aufgesetzt wird, bezeichnet man diese Verbindung als TCP/IP-Protokoll.
(vgl. http://de.wikipedia.org/wiki/Transmission_Control_Protocol, 09.05.2006)

URI (Uniform Resource Identifier): „ Ein Uniform Resource Identifier (URI) (engl. „einheitlicher Bezeichner für Ressourcen)" ist eine Zeichenfolge, die zur Identifizierung einer abstrakten oder physikalischen Ressource dient. URIs werden zur Bezeichnung von Ressourcen (wie Webseiten, sonstigen Dateien, Aufruf von Webservices, aber auch z. B. E-Mail-Empfängern) im Internet und dort vor allem im WWW eingesetzt."
(http://de.wikipedia.org/wiki/URI, 09.05.2006)

User-Agent: „Ein User Agent ist ein Client Programm, mit dem ein Netzwerkdienst genutzt werden kann. Der User Agent ist die Schnittstelle zum Benutzer, die die Inhalte darstellt und Befehle entgegennimmt. Beispiele für User Agents sind Webbrowser, E-Mail-Programme, Newsreader und IRC-Clients."
(http://de.wikipedia.org/wiki/User_Agent, 09.05.2006)

Web Server: Ein Web Server kann Webseiten, Servlets oder auch Web Services anbieten.

WS-I: Die Web Service Interoperability Organisation ist ein Zusammenschluss mehrerer großer Firmen, die darum bemüht sind, Web Services über alle Plattformen hinweg interoperabel zu machen. Mit dem Basic Profile Version 1.0 ist 2004 der erste Standard des WS-I erschienen.

9. Quellenverzeichnis

9.1. Bücher

„ASP.NET in a nutshell", G. Andrew Duthie, Matthew MacDonald, Second Edition, 2003, O'Reilly; ISBN 0-596-00520-2

"CSharp in a Nutshell", Ben Albahari, Peter Drayton, Ted Neward, Second Edition, 2003, O'Reilly; 0-596-00526-1

"Java and SOAP", Robert Englander, 2002, O'Reilly; ISBN 0-596-00175-4

„Programming Web Applications with XML-RPC", Simon St. Laurent, Joe Johnston, Edd Dumbill, First Edition, 2001, O'Reilly; ISBN 0-596-00119-3

„Programming Web Services with SOAP", Doug Tidwell, James Snell, Pavel Kulchenko, First Edition, 2001, O'Reilly; ISBN 0-596-00095-2

„Web Services - Die Standards", Tobias Hauser, Ulrich M. Löwer, 2003, Galileo Press; ISBN 389842393X

„XML Web Service", Christian Weyer, 2002, Addison-Wesley Verlag; ISBN 3-8273-1891-2

„Web Service Essentials", Ethan Cerami, First Edition, 2002, O'Reilly; ISBN 0-596-00224-6

9.2. Manuskripte

„Einführung ins wissenschaftliche Arbeiten und Forschungsmethoden", Mag. Edith Huber, 2005, Fachhochschule St. Pölten

9.3. Internet

9.3.1. Spezifikationen

DCOM
http://www.webopedia.com/TERM/D/DCOM.html

W3C veranstaltet Seminar zu Web-Services-Semantik
Tikwinski, http://www.w3c.de/Press/2006/wss-pressrelease.de.html

RPC
http://www.linuxfibel.de/rpc.htm
http://www.linux-magazin.de/Artikel/ausgabe/2003/06/024_news_insec/insec_news.html

SOAP
Computerwoche.de, http://www.computerwoche.de/541408
msdn.microsoft.com,
http://msdn.microsoft.com/webservices/webservices/understanding/webservicebasics/default.aspx?pull=/library/en-us/dnwebsrv/html/rpc_literal.asp
Php Magazin,
http://www.phpmagazin.de/itr/online_artikel/psecom,id,649,nodeid,62.html

Stefan Tilkov's Webblog,
http://www.innoq.com/blog/st/2003/03/web_servicesstile_dokument_vs_rpcorientiert.html

UDDI
http://www.uddi.org/about.html

Wsdl.exe
http://www.microsoft.com/germany/msdn/library/web/WebServicesMitNETFramework20UndVisualStudio2005.mspx?mfr=true

XML-RPC
http://www.xmlrpc.com/
http://www.xmlrpc.com/spec

XML-RPC for PHP
http://phpxmlrpc.sourceforge.net/

"Middleware heute", Andreas Kapp, 2003,
http://www.pythonfactory.com/webapps/default/space/Middleware+heute

9.3.2. Wikipedia

CORBA
http://de.wikipedia.org/wiki/CORBA

EAI
http://de.wikipedia.org/wiki/Enterprise_Application_Integration

Host
http://de.wikipedia.org/wiki/Host_%28Informationstechnik%29

RMI
http://de.wikipedia.org/wiki/Remote_Method_Invocation

Schnittstellen
http://de.wikipedia.org/wiki/Schnittstelle

SOAP
http://de.wikipedia.org/wiki/SOAP

TCP
http://de.wikipedia.org/wiki/Transmission_Control_Protocol

9.3.3. Quellen der Implementierungen

Java – XML-RPC
Apache XML-RPC Source + Tutorial: http://ws.apache.org/xmlrpc/
Rox Source + Tutorial: http://rox-xmlrpc.sourceforge.net/

.NET – XML-RPC
XML-RPC.NET Source + Tutorial: http://www.xml-rpc.net/
XmlRpcCS Source: http://xmlrpccs.sourceforge.net/

PHP – XML-RPC
XML-RPC for PHP Source: http://phpxmlrpc.sourceforge.net/
XML-RPC for PHP Tutorial: http://www.zend.com/zend/tut/tut-xml-rpc-client.php
IXR Source + Tutorial: http://scripts.incutio.com/xmlrpc/

Java – SOAP
Apache SOAP Source: http://ws.apache.org/soap/index.html
Apache SOAP Tutorial: http://www.soapuser.com/server1.html
Apache Axis Source: http://ws.apache.org/axis/
Apache Axis Tutorial: http://www.torsten-horn.de/techdocs/java-soap-axis.htm

.NET – SOAP
.NET Framework: http://msdn.microsoft.com/netframework/downloads/updates/
ASP.NET Tutorial: http://www.aspheute.com/artikel/20010621.htm

PHP – SOAP
nuSOAP Source: http://sourceforge.net/projects/nusoap/
nuSOAP Tutorial: http://www.zend.com/zend/tut/tutorial-campbell.php
PHP5 SOAP Source: http://at.php.net/manual/de/ref.soap.php
PHP5 SOAP Tutorial: http://devzone.zend.com/node/view/id/689

10. Abbildungsverzeichnis

11. Codeverzeichnis

12. Danksagung

Mein erster Dank gebührt meinen Diplomarbeitsbetreuern DI Grischa Schmiedl und DI (FH) Fritz Grabo die sich beide sehr große Mühe gegeben haben, mich bei meiner Diplomarbeit zu unterstützen, sodass ich auf das Endresultat wirklich stolz sein kann.

Ein besonderer Dank gebührt meiner Mutter Susanne, die mich nicht nur als kleiner Junge in genau dem richtigen Maße zu guten Leistungen antreiben konnte, sondern mir darüber hinaus auch das Studium finanziert hat, welches zu dieser Arbeit führte.

Bei den beiden Menschen, die meine Arbeit Korrektur gelesen haben, nämlich Christine Labes und Bojan Sadjak, möchte ich mich ebenfalls ganz herzlich bedanken, denn wer die Rohfassungen gekannt hat weiß, dass das keine leichte Aufgabe war.

Abschließend möchte ich noch den zahlreichen Menschen danken, die an der Entwicklung und Verbreitung des Internets beteiligt waren, denn ohne diese Personen wäre diese Diplomarbeit, weder inhaltlich noch technisch, möglich gewesen.

Kurz gesagt: **Vielen Dank!**