DIPLOMARBEIT

Vergleich von SOAP Kommunikationsplattformen

ausgeführt am

Institut für Programmiersprachen
der Technischen Universität Wien

unter der Anleitung von
Ao.Univ.Prof. Dipl.-Ing. Dr. Franz Puntigam

durch

Manfred Jakesch
Gartengasse 15
2443 Deutsch-Brodersdorf

Wien, 14. Oktober 2004 _____

Kurzfassung

Das SOAP Protokoll wurde als ein plattformunabhängiges Nachrichtenformat definiert. Aber alleine mit dieser Definition ist diese Unabhängigkeit nicht erreicht. Es gibt immer noch Rahmenbedingungen, die Einschränkungen dieser Eigenschaft beinhalten. In dieser Arbeit soll nun für SOAP Kommunikationsplattformen aus Java und .NET die Umsetzung der SOAP Spezifikation näher untersucht werden. Dabei ist die Interoperabilität der SOAP Implementierungen ein entscheidender Faktor. Nur mit der Möglichkeit, auch leicht über Plattformgrenzen hinweg Daten austauschen zu können, ist der Siegeszug von SOAP fortzusetzen. Dabei bietet SOAP nicht nur Mittel des reinen Datenaustauschs. Es können auch verschiedenste zusätzliche Funktionen durch die leichte Erweiterbarkeit dieses Protokolls implementiert werden. Diesen zusätzlichen SOAP Features ist ein weiterer Teil dieser Arbeit gewidmet. Es wird gezeigt, welche Funktionen im Moment bei den einzelnen untersuchten Implementierungen zur Verfügung stehen, und ob und wie sie plattformübergreifend genutzt werden können.

Abstract

The SOAP Protocol has a platform independent message format. However, the best design is not useful if this independence is not implemented at all or not implemented correctly. There are certain criteria that limit the possibilities of SOAP. This thesis tries to compare SOAP implementations of Java and the .NET framework. Interoperability of SOAP implementations is therefore a big issue for this work. The success of the SOAP protocol can only be prolonged if there is a way to send data across different platforms and programming languages. But SOAP is not limited to sending only data. Because of the SOAP Extensibility feature it is possible to add many additional functions that are not part of the SOAP recommendation. These SOAP Features build another part of this work. In this thesis you will find a comparison of the features that are provided by each implementation. We will answer the question whether these features can be used across different SOAP platforms.

Danksagung

An erster Stelle möchte ich mich hier bei meinen Eltern Theresia und Manfred bedanken, ohne deren Hilfe und Rückhalt diese Arbeit nicht entstehen hätte können. Mein besonderer Dank gilt auch meinem Neffen Fabian.
Bei dieser Gelegenheit möchte ich mich bei Professor Franz Puntigam noch recht herzlich bedanken, dass er mir bei diversesten Fragen hilfreich zur Seite gestanden ist.

Inhaltsverzeichnis

1 Einleitung

SOAP ist bereits ein integraler Bestandteil der Web Service Architektur von heute ([33]). Es ist bei der Erstellung von Applikationen, die sich auf verteilte Systeme stützen, nicht mehr wegzudenken. Diese Arbeit beschäftigt sich mit der Umsetzung des SOAP Protokolls auf Seiten der Programmiersprachen, und welche Funktionalitäten diese den Applikationen zur Verfügung stellen.

1.1 Motivation

Grundlage für diese Arbeit stellt die in der Praxis vorkommende Notwendigkeit dar, Daten zwischen verschiedenen Plattformen möglichst effizient und sicher zu übertragen. Meist soll aber auch eine gewisse Semantik der Daten mit übertragen werden, die es den auf diesen Daten operierenden Applikationen erlaubt, geordnet und möglichst typsicher auf die empfangenen Strukturen zuzugreifen. Dies bedeutet, dass die empfangenen Daten möglichst genau auf bestimmte Datentypen der jeweiligen Programmiersprache umgesetzt werden sollen. SOAP bietet diese Möglichkeit des Datenaustauschs. Da dieser SOAP-gestützte Nachrichtenaustausch aber über Plattformgrenzen hinweg stattfinden kann, ist nicht immer sichergestellt, dass die Programmiersprache, in der die Daten versandt wurden, auch an der Empfängerseite und somit das gleiche Typsystem verwendet wird. Hier kommt der Begriff der Interoperabilität ins Spiel, der auch einen großen Einfluss auf diese Arbeit hat. Es sollen nämlich anhand von ausgesuchten SOAP Kommunikationsplattformen Interoperabilitätsprobleme beschrieben beziehungsweise aufgedeckt werden. Ziel ist es, einen Überblick über die derzeitige Lage zu schaffen.
Diese Technologie wird auch sehr oft im Bereich von Web Services angewandt, sodass SOAP und Web Services oft fälschlicherweise als synonyme Begriffe verwendet werden. Hier ist es vor allem wichtig, dass eine plattformunabhängige Methode zur Kommunikation zwischen verschiedenen Komponenten benützt werden kann. Es soll in dieser Arbeit aber auch gezeigt werden, dass SOAP nicht nur mit Web Services verwendet wird. Als Beispiel sei jetzt nur .NET Remoting genannt, dass auf SOAP Nachrichten zur Serialisierung von Objekten aufbaut und nicht ursächlich für das Erstellen von Web Services konzipiert wurde.
Nun gilt es zunächst zu untersuchen, ob in den Implementierungen der bekannten höheren Programmiersprachen der SOAP Version 1.2 Standard, welcher in der Spezifikation vom World Wide Web Konsortium (kurz W3C) festgelegt wurde, auch wirklich Umsetzung gefunden hat. Beim Entwurf dieser Spezifikation wurde besonders auf gewisse Probleme der Interoperabilität der Vorversionen Rücksicht genommen und versucht, diese Problemstellen wei-

testgehend auszumerzen. Probleme bezüglich Interoperabilität der einzelnen SOAP Kommunikationsplattformen sind nämlich eines der größten Hindernisse bei der Kommunikation mittels SOAP und scheinen durch immer ausgefeiltere Anforderungen an die SOAP Implementierungen stetig vielfältiger zu werden.

1.2 Zielsetzung

Ziel dieser Arbeit ist es, einen Überblick über die drei untersuchten SOAP Kommunikationsplattformen zu geben. Es stehen vor allem die Interoperabilität der untersuchten Implementierungen und die technische Umsetzung von SOAP Features im Mittelpunkt. Es soll aber ebenso auf andere SOAP Implementierungen eingegangen beziehungsweise auf das Problem mit der Interoperabilität an sich hingewiesen werden.
Zum Punkt der Interoperabilität wäre noch anzumerken, dass, so es im Bereich des Möglichen ist, auch Lösungsansätze zur Umgehung dieser Problematik aufgezeigt werden sollen. Ebenso ist es ein Ziel, zu zeigen, wo Interoperabilitätsprobleme durch Ungenauigkeiten in der Spezifikation zu SOAP 1.1 aufgetreten sind, und wie diese dann in der Nachfolgeversion behoben worden sind.

1.3 Durchführung

Ein großer Teil der Vorbereitung dieser Arbeit bestand aus dem Finden und Studieren der notwendigen Literatur. Da es sich beim Thema „Vergleich von SOAP Kommunikationsplattformen“ um ein sehr aktuelles Thema handelt, war eines der Hauptmedien zur Beschaffung von Information das Internet. Dort sind nämlich sehr viele technische Artikel zu diesem Thema zu finden, die die Grundlage für diese Arbeit und den dazugehörigen Vergleich bilden. Außerdem handelt es sich bei SOAP um einen Standard, der sich aus den Technologien des World Wide Web gebildet hat ([19]). So ist es nur natürlich, hier nach Informationen zu diesem Thema zu suchen.
Um die mittels Literaturrecherche identifizierten Probleme beziehungsweise Anwendungsbeispiele für die diversen SOAP Implementierungen näher studieren zu können, war es notwendig, Testumgebungen für die einzelnen SOAP Kommunikationsplattformen aufzusetzen. Mit diesen Testinstallationen war es möglich, die erkannten Interoperabilitätsprobleme beziehungsweise Implementierungsdetails näher zu beleuchten. Dies half vor allem, die Einsicht in die Problematik zu stärken. Ein weiteres Aufgabengebiet dieser Tests war die Überprüfung, welche SOAP Features konkret mittels der jeweiligen SOAP Implementierung realisiert werden können. Dies ist vor allem bei

den Implementierungen notwendig gewesen, deren Dokumentation nicht die gewünschte Ausführlichkeit hatte, oder die untersuchten Features sich noch im Entwicklungsstadium befanden.
Da diese Arbeit auch sehr stark mit den Standards verknüpft ist, die dem SOAP Protokoll zugrunde liegen, war das Studium einiger Spezifikationen beziehungsweise Recommendations, wie diese Dokumente beim WWW Konsortium heißen, ein wichtiger Punkt für das Gelingen dieser Arbeit. Diese Spezifikationen schaffen einen guten Überblick über das Themengebiet und bieten die Ausgangslage für die Entwicklerteams der einzelnen SOAP Kommunikationsplattformen. Interessant ist, dass einige Entwickler der SOAP Spezifikation auch aktiv bei SOAP Implementierungen mitgewirkt haben.

1.4 Aufbau dieser Arbeit

Es ist von Nöten, die Grundlagen dieses Protokolls zu erläutern. Dies geschieht im Abschnitt 2. Das Hauptaugenmerk wird in dieser Arbeit auf die SOAP Version 1.2, die Spezifikation ist in [39] zu finden, gelegt. Schwerpunkt bei der Erstellung der aktuellen Spezifikation war eine Erhöhung der Interoperabilität. Die Spezifikation ist natürlich als aktuellste Version von SOAP von höchster Relevanz. Dieser Abschnitt enthält einen Unterabschnitt über die Unterschiede zwischen SOAP 1.2 und dessen Vorgängerversion. Die Gründe, warum dieses Kapitel schlussendlich in diese Arbeit gelangt ist, werden später noch näher erläutert.
Ziel dieses Abschnitts ist es, einen Überblick über die Möglichkeiten des SOAP Protokolls zu geben. Dies ist für die späteren Abschnitte relevant, wenn untersucht werden soll, welche dieser vorgestellten SOAP Features ihre Umsetzung in den untersuchten SOAP Implementierungen gefunden haben. Außerdem ist es notwendig, ein gemeinsames Vokabular zu erarbeiten, auf dass in späteren Kapiteln wieder zurückgegriffen werden kann, um konkrete Einzelheiten einer Implementierung näher beleuchten zu können.
Bei den Programmiersprachen möchte ich mich hier auf zwei höhere Programmiersprachenumgebungen beschränken. Es sollen die meistverwendeten Implementierungen aus Java und der .NET Plattform untersucht werden. Diese Programmiersprachen beziehungsweise Entwicklungsumgebungen stellen auch den größten Teil der Realisierungsplattformen für aktuelle Services dar und sind deshalb, auch was die Interoperabilität dieser SOAP Implementierungen betrifft, von großem Interesse ([64]).
Konkret ist die Wahl auf das Apache Axis Projekt, die von Sun unterstützte Java XML RPC Referenzimplementierung und die beiden SOAP Kommunikationsplattformen des .NET Frameworks, nämlich die ASP.NET Umgebung und .NET Remoting, gefallen. Jede dieser SOAP Kommunikations-

plattformen wird eingehend untersucht werden. Die Architektur sowie die Hauptmerkmale jeder einzelnen Implementierung werden in einem eigenen Abschnitt ausführlicher behandelt werden.
Ein weiteres Kapitel dieser Arbeit beschäftigt sich mit dem Vergleich der oben genannten SOAP Kommunikationsplattformen. Eine Untersuchung der konkreten Umsetzung der SOAP 1.2 Spezifikation war einer der Punkte, die es zu berücksichtigen galt. Ebenso ist eine ausführliche Erläuterung über die Probleme mit der Interoperabilität von SOAP zu finden. Es wird stark auf die Auswirkungen dieser Probleme auf die drei untersuchten SOAP Kommunikationsplattformen eingegangen. Es sollen aber auch Lösungswege aufgezeigt werden, die einen Ausweg aus dieser Misere darstellen könnten. Ein ausführlicherer Abschnitt ist den Problemen mit den Typsystemen gewidmet. Dort wird auch die Problematik der plattformabhängigen Typen und deren Umsetzung in die XML-Datentypen näher beleuchtet werden.
Beim Vergleich der einzelnen SOAP Implementierungen ist mir wichtig, die konkreten Umsetzungen von SOAP Features in den SOAP Kommunikationsplattformen miteinander zu vergleichen. Es kommt vor allem darauf an, welche Mittel in der konkreten Implementierung überhaupt zur Verfügung stehen. Ein zweiter wichtiger Punkt ist, welche Technologie hinter dem umgesetzten SOAP Feature steht, und wie diese Technologie dann mit den anderen Umsetzungen korrespondiert.
Die Sicherheit von Services, die auf SOAP basieren, ist ein weiteres Thema dieser Arbeit. Es gilt zu untersuchen, welche Mechanismen die SOAP Kommunikationsplattformen zur Verfügung stellen, um Vertraulichkeit, Integrität und Datensicherheit zu garantieren.
In dem folgenden Abschnitt soll das SOAP Protokoll genauer untersucht werden. Dieses Kapitel bildet den idealen Einstieg in die Welt von entfernten Services und SOAP.
Am Ende dieser Arbeit findet sich eine Zusammenfassung, in der alle gesammelten Eindrücke noch einmal Revue passieren werden.

2 SOAP

Dieses Kapitel ist der SOAP Version 1.2 gewidmet. Zusätzlich wird auch ein Vergleich mit der Vorversion angestellt, um eventuelle Schwachpunkte in der SOAP 1.1 Spezifikation später näher erläutern zu können.
Das W3C hat SOAP in [39] wie folgt beschrieben:

> SOAP Version 1.2 (SOAP) ist ein leichtgewichtiges Protokoll für das Austauschen strukturierter Information in einer dezentralisierten, verteilten Umgebung. Dabei stützt es sich auf XML-Technologien zur Definition eines erweiterungsfähigen Nachrichten-Frameworks. Es stellt ein Nachrichtenkonstrukt zur Verfügung, das auf einer Reihe von Transportprotokollen aufsetzen kann. Das Framework wurde entworfen, um unabhängig von spezifischen Programmiermodellen und anderer implementationsspezifischer Semantik zu sein.

Es haben sich die Einfachheit und Erweiterbarkeit als die zwei Hauptdesignziele herauskristallisiert.
Um die zu untersuchenden Implementierungen auf die Unterstützung der SOAP Spezifikation und der gegebenen Interoperabilität testen zu können, ist es notwendig, auf die Feinheiten des Protokolls näher einzugehen. Zum Abschluss dieses Abschnitts soll dann noch die Rolle der WSDL (Web Service Description Language) kurz erläutert, und der bereits oben erwähnte Vergleich der beiden SOAP Versionen präsentiert werden.

2.1 Die historische Entwicklung

SOAP wurde mit dem Ziel entwickelt, XML-basierte Methoden-Aufrufe über Netzwerk zu ermöglichen. Deshalb stand zu Anfang das Akronym SOAP für Simple Object Access Protocol. Seit seiner ursprünglichen Spezifikation hat dieses Protokoll schon mehrere Evolutionsstufen durchlebt.
Die originale Spezifikation war SOAP 0.88, die als Urform der aktuellen SOAP-Spezifikation gezählt werden kann. Davor gab es eigentlich nichts Vergleichbares. Es gab zwar mehrere XML-basierte Protokolle, die auf Nachrichtenaustausch beziehungsweise Remote Procedure Calls ausgelegt waren, die es aber nie zu größerer Verbreitung gebracht hatten ([7]).
Im Frühjahr des Jahres 2000 war es dann soweit. Mehrere Entwickler von Microsoft, IBM, DevelopMentor, Lotus Development Corporation und UserLand Software arbeiteten an einer neuen Version von SOAP. Unter diesen Entwicklern wären die bekannten SOAP Gründer wie Dave Winer, Don Box oder auch Henrik Frystyk Nielsen zu erwähnen. Es entstand die Version 1.1

des Protokolls. Hier ist anzumerken, dass diese SOAP Spezifikation zwar beim W3C aufliegt, die Kommission aber keine Kontrolle und Möglichkeit des Änderns dieses Dokuments hat ([25]).
Diese Version war ähnlich der Vorgängerversion SOAP 1.0, aber das HTTP Binding und ein paar andere SOAP Features wurden runderneuert. Wichtig war noch, dass man SOAP aus dem Eck der Remote Procedure Calls herausgeholt hat, und es nun auch möglich war, dieses Protokoll für andere Kommunikationsmodelle zu nutzen. Man hatte sich aber etwas von den vorherigen Versionen abgesetzt, da diese als einzige Anwendung die RPCs kannten. In der SOAP Version 1.1 ist es noch immer möglich, Remote Procedure Calls abzusetzen. Diese sind aber nicht mehr die einzige Möglichkeit, um SOAP für den Nachrichtenaustausch zu verwenden.
Mittlerweile ist die Spezifikation zu Version 1.2 schon fertig gestellt. Es gilt festzuhalten, dass es sich bei dieser Spezifikation um eine wirkliche Recommendation des WWW Konsortiums handelt. Diese Version der SOAP Spezifikation wurde nämlich von der XML Protocol Working Group verfasst. Alle Änderungsrechte liegen bei dieser Gruppe. Dies ist einer der Unterschiede zu den Vorversionen von SOAP, wo ja einzelne Firmen die Oberhand über die Spezifikation hatten.
In dieser Version ist man noch einen Schritt weiter gegangen als bei der eben besprochenen Version 1.1. Hier wurde das RPC-Modell noch weiter in Frage gestellt, indem man die zugehörige SOAP-Kodierung für optional erklärt hat. Eine Implementierung, die versucht SOAP 1.2 konform zu sein, muss dieses Kodierungsschema nicht unbedingt umsetzen. Bei dem erwähnten Schema handelt es sich um ein Überbleibsel aus den früheren SOAP Versionen. Da es zur Zeit der ersten Spezifikationen noch keine XML Schema Spezifikation gab, beziehungsweise diese sich noch in keinem produktreifen Zustand befanden, wurde eine Möglichkeit gesucht, wie Daten einer Programmiersprache in ein XML-Format überführt werden können. Aus diesem Grund wurde mit der SOAP-Kodierung ein Regelwerk erstellt, dass helfen sollte, diese Daten in eine XML-Repräsentation abzubilden. Da aber mittlerweile die XML Schema Spezifikation vollendet wurde und sich als sehr stabil und umfassend erwiesen hat, wird in näherer Zukunft der Umstieg auf diese XML-Daten-Beschreibungssprache erfolgen. Teilweise ist der Umstieg bei einigen bekannten Implementierungen schon erfolgt ([30]).

2.2 Allgemeines

Die in dieser Arbeit verwendete Namensraumabkürzung „*xsd*“ steht ausnahmslos für den Namensraum der aktuellen XML Schema Spezifikation mit der URI „`http://www.w3.org/2001/XMLSchema`“. Alle erwähnten XML

Schema Typen sind aus diesem Namensraum und beziehen sich somit auf die Spezifikation aus [12].

2.3 SOAP Knoten und Rollen

Nun wollen wir uns mehr den theoretischen Grundlagen von SOAP widmen. In den folgenden Abschnitten soll der Aufbau einer SOAP Infrastruktur und der dazugehörigen Nachrichten näher untersucht werden.
Ein SOAP Knoten kann ein initialer SOAP Sender, ein ultimativer SOAP Empfänger oder ein SOAP Intermediary sein. Ein initialer SOAP Sender ist ein Knoten, der eine SOAP Nachricht erzeugt und diese an einen anderen SOAP Knoten weiterreicht. Bei einem ultimativen SOAP Empfänger handelt es sich um einen Knoten, der eine SOAP Nachricht empfängt und diese dann bearbeitet. Er kann eine neue Nachricht erstellen, darf aber die empfangene Nachricht nicht weitersenden. Da SOAP für ein verteiltes System konzipiert wurde, kann es auch vorkommen, dass eine SOAP Nachricht von einem Knoten empfangen wird, der sie nach einer eventuellen Verarbeitung an einen anderen Knoten weitersendet. Einen solchen Knoten würde man laut SOAP Spezifikation Intermediary nennen. Jeder SOAP Knoten muss, der eine SOAP Nachricht empfängt, diese nach dem SOAP Processing Model verarbeiten. Näheres zum SOAP Abarbeitungsmodell findet man in Abschnitt 2.5.
Beim Verarbeiten spielt der SOAP Knoten eine oder mehrere sogenannte Rollen. Jede von diesen Rollen wird durch eine URI (kurz für Uniform Resource Identifier) identifiziert, die man auch den SOAP Rollen Namen (SOAP role name) nennt.
Mit Ausnahme der drei in Tabelle 1 definierten SOAP Rollennamen schreibt die Spezifikation nicht vor, an welchen Kriterien man erkennt, welche Rolle ein Knoten zu spielen hat, wenn er eine bestimmte Nachricht erhält. Es bleibt den Implementierungen offen, ob sie etwa hart kodierte Einstellungen in der Implementation vorsehen, Informationen im Transportprotokoll zur Verfügung stellen oder etwa Konfigurationseinstellungen während der Installation durch einen Benutzer zulassen.

2.4 Aufbau einer SOAP Nachrichten

Zwischen zwei oder mehreren SOAP Knoten werden sogenannte SOAP Nachrichten (SOAP Messages) ausgetauscht. Eine SOAP Nachricht ist nach [39] ein XML Infoset mit einem Document Information Item, das exakt ein Kindelement enthält, das ein SOAP `Envelope` Element Information Item sein muss.

Kurzname	Name	Beschreibung
`next`	http://www.w3.org/2003/05/soap-envelope/role/next	Jeder SOAP Intermediary und der ultimative Empfänger müssen in der Lage sein, diese Rolle zu übernehmen.
`none`	http://www.w3.org/2003/05/soap-envelope/role/none	Es wurde keine spezifische Rolle definiert.
`ultimativeReceiver`	http://www.w3.org/2003/05/soap-envelope/role/ultimativeReceiver	Dieser Rollenname definiert einen ultimativen Empfänger. Jeder SOAP Knoten, der als ein Empfänger einer Nachricht auftritt, aber diese dann nicht weitersendet, muss in der Lage sein, diese Rolle zu übernehmen.

Tabelle 1: SOAP Rollen definiert in der Spezifikation [39]

Eine SOAP Nachricht besteht also laut Spezifikation aus einem XML Infoset. Ein wohlgeformtes XML Dokument stellt eine Serialisierung dieser Darstellungsform dar. Da XML Dokumente die gebräuchlichste Ausformung von XML Infosets sind, wird diese Darstellungsform auch ausnahmslos in dieser Arbeit verwendet werden, wenn es um die Darstellung von SOAP Nachrichten beziehungsweise deren Elementen geht.
Listing 1 zeigt eine beispielhafte SOAP Nachricht. Es handelt sich bei diesem Beispiel um eine fiktive Reservierung eines Hotelzimmers und eines Autos. An diesem Beispiel zeigt sich der typische Aufbau einer SOAP Nachricht, wie er auch noch in Abbildung 1 schematisch dargestellt ist. Die dargestellte Nachricht besteht aus zwei sogenannten SOAP Header Blöcken, Abschnitt 2.4.2 ist diesem SOAP Nachrichten Element gewidmet, und einem SOAP Body. Näheres zum Thema SOAP Body findet man in Abschnitt 2.4.3.

2.4.1 SOAP Envelope

Ein SOAP `Envelope` Element Information Item hat folgende definierte Eigenschaften:

- einen lokalen Namen `Envelope`
- einen Namensraum „http://www.w3.org/2003/05/soap-envelope“

```
<?xml version='1.0' ?>

<soap:Envelope xmlns:soap="http://www.w3.org/2003/05/soap-envelope">
  <soap:Header>
    <m:Reservierung
      xmlns:m="http://manfred.jakesch.at/Beispiel/Reservierung"
      soap:role="http://www.w3.org/2003/05/soap-envelope/role/next"
      soap:mustUnderstand="true">
      <m:Referenz>2658302-5568-mfj</m:Referenz>
      <m:Datum>24.05.2004 16:35</m:Datum>
    </m:Reservierung>
    <n:Gast
      xmlns:n="http://manfred.jakesch.at/Beispiel/Gast"
      soap:role="http://www.w3.org/2003/05/soap-envelope/role/next"
      soap:mustUnderstand="true">
      <n:Name>Manfred Jakesch</n:Name>
    </n:Gast>
  </soap:Header>
  <soap:Body>
    <h:Hotel
      xmlns:h="http://manfred.jakesch.at/Beispiel/Gast">
      <h:Zimmer>239</h:Zimmer>
      <h:Von>27.09.2004</h:Von>
      <h:Bis>03.10.2004</h:Bis>
    </h:Hotel>
    <a:Autoverleih
      xmlns:a="http://manfred.jakesch.at/Beispiel/Auto">
      <a:Auto>Honda Civic</a:Auto>
      <a:Kennzeichen>BN 235 LS</a:Kennzeichen>
    </a:Autoverleih>
  </soap:Body>
</soap:Envelope>
```

Listing 1: eine SOAP Nachricht

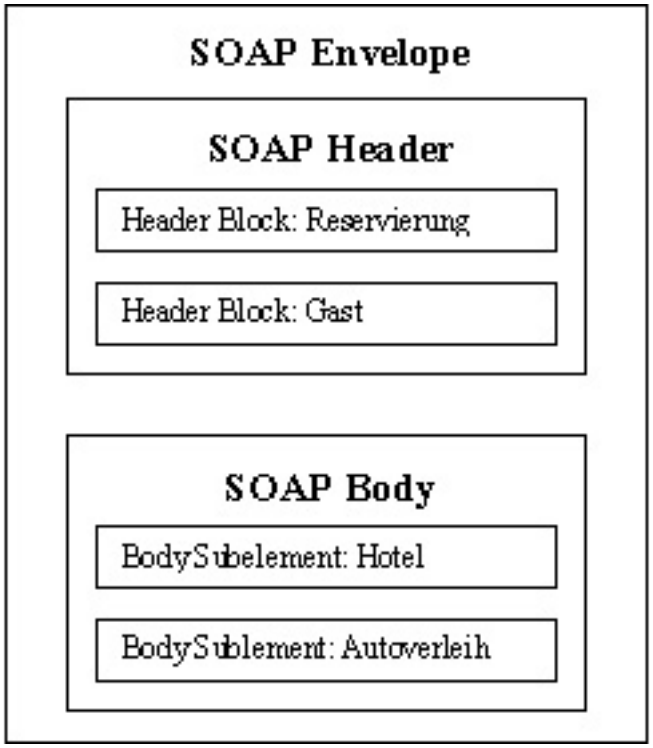

Abbildung 1: Schematische Darstellung der SOAP Nachricht aus Listing 1

- beliebig viele namensraumqualifizierte Attribute Information Items unter seinen Attributen. Mit namensraumqualifiziert ist hier gemeint, dass das Attribute Information Item ganz klar einem XML Namensraum zugeordnet werden können muss.
- einen oder zwei Element Information Items als seine Kinder in der folgenden Reihenfolge:
  1. ein optionales `Header` Element Information Item
  2. genau ein SOAP `Body` Element Information Item

Jedes Kindelement eines SOAP Envelopes kann ein Attribut `encodingStyle` enthalten, das dazu verwendet werden kann, einer Applikation anzuzeigen, wie die Daten, die dieses Element enthält, deserialisiert werden können. Es bleibt aber anzumerken, dass dieses Attribut als optional zu behandeln ist, es also nicht zwingendermaßen angegeben werden muss. Die Kindelemente eines übergeordneten Elements erben dieses Attribut, sofern sie kein eigenes `encodingStyle` Attribut definiert haben.

Das `encodingStyle` Attribute Information Item hat den Typ *xsd:anyURI*. Der Wert dieses Attributs identifiziert eine Menge von Deserialisierungsregeln, die benutzt werden können, um die SOAP Nachricht ordnungsgemäß zu deserialisieren.

2.4.2 SOAP Header

Ein SOAP Header dient zur Erweiterung einer SOAP Nachricht. Darin können verschiedenste nicht unbedingt applikationsspezifische Daten enthalten sein. Außerdem sind SOAP Header nützlich, um etwa Daten zu transportieren, die nicht direkt von dem empfangenden Modul verarbeitetet werden können. Dies könnten zum Beispiel Security Informationen oder eine Transaktions-ID sein.
Interessant ist die Verbindung von SOAP Headern mit der Rolle *none*. Man nehme einmal an, es wurde ein Service entworfen, das nicht wirklich erweiterungsfähig ist. Die neueste Version liefert beziehungsweise verarbeitet nun aber auch Daten, die über die der alten Version hinausgehen. Eine Veränderung der Schnittstelle ist in solchen Fällen meistens nicht durchzuführen und vielleicht auch gar nicht wünschenswert. Eine echte Alternative bietet ein SOAP Header mit der Rolle *none*. Mittels einem solchen Nachrichtenelements ist es möglich, reine Applikationsdaten im Header Block zu übertragen.
Ein `Header` ist wie folgt aufgebaut:

- ein lokaler Name `Header`
- er liegt im Namensraum „http://www.w3.org/2003/05/soap-envelope“
- beliebig viele Attribute, die aber namensraumqualifiziert sein müssen
- beliebig viele Kindelemente, die ebenfalls namensraumqualifiziert sein müssen

Jedes Kindelement des SOAP Headers wird auch als SOAP Header Block bezeichnet.
Solch ein Header Block darf entweder Zeichen beinhalten oder aber eine nicht näher spezifizierte Anzahl an Kindelementen enthalten. Außerdem darf er auch beliebig viele Attribute enthalten. Diese Attribute können unter anderem die folgenden sein:

- `encodingStyle` Attribut: Dieses Attribut wurde bereits oben näher erläutert. Es dient dazu, Deserialisierungsinformation für die Datenelemente des Header Blocks dem Empfänger einer SOAP Nachricht bereitzustellen.

- `role` Attribut: Es enthält die Information, für welchen SOAP Knoten dieser SOAP Header Block bestimmt ist. Die Rollen, die ein SOAP Knoten einnehmen kann, wurden bereits in Abschnitt 2.3 untersucht. Wird dieses Attribut nicht gesetzt, so muss angenommen werden, dass dieser Informationsblock für den ultimativen Empfänger bestimmt ist. Es muss berücksichtigt werden, dass dieses Attribut nur für SOAP Header Blöcke gesetzt werden darf, denn laut Spezifikation müssen Empfänger von SOAP Nachrichten dieses Attribut in Kindelementen von SOAP Header Blöcken ignorieren.

- `mustUnderstand` Attribut: Dieses Attribut zeigt dem SOAP Empfänger an, ob er diesen SOAP Header Block bearbeiten muss, oder ob er ihn als optional behandeln kann. Es ist vom Typ *xsd:boolean*, kann also „true“ oder „false“ sein. Wird dieses Attribut nicht explizit gesetzt, so ist der Wert als „false“ anzunehmen. Näheres zu diesem Attribut wird im Abschnitt 2.5 erläutert werden.

- `relay` Attribut: Mit diesem Attribut kann einem SOAP Empfänger angezeigt werden, ob der damit in Verbindung gebrachte SOAP Header Block weitergeleitet und nicht weiter bearbeitet werden soll. Auch dieses Attribut ist vom Typ *xsd:boolean*. Es gilt die gleiche Einschränkung wie für das Attribut `role`, dass es nur in Verbindung mit einem Header Block gesetzt werden darf. Ein SOAP Empfänger sollte dieses Attribut in Kindelementen eines Header Blocks ignorieren. Ist dieses Attribut in einem SOAP Header Block nicht enthalten, so wird laut Spezifikation der Wert „false“ dafür angenommen.

Listing 2 zeigt an einem Beispiel ein SOAP Header Element mit einem Headerblock namens Transaktion, dem Wert 79 und dem Beispielnamensraum für dieses Element mit „*http://manfred.jakesch.at/Beispiel/SOAPHeader*“. Dieser Beispiel-Header könnte dazu dienen, um im Applikationsrahmen einen Transaktionskontext aufzubauen, der aber von SOAP nicht direkt vorgesehen ist. Dieses Beispiel zeigt sehr schön, wie modular sogenannte SOAP Features entwickelt werden können. Mit der einfachen Erweiterbarkeit des SOAP Protokolls wird hier eine der Stärken demonstriert. Für eben diese Erweiterungen der SOAP Kommunikation stellen die SOAP Header einen sehr wichtigen Mechanismus zur Verfügung.

2.4.3 Der SOAP Body

Der SOAP Body einer SOAP Nachricht enthält die eigentlichen applikationsspezifischen Daten, die direkt von der Applikation verarbeitet werden können.

```
<env:Header
  xmlns:env="http://www.w3.org/2003/05/soap-envelope">
  <t:Transaktion
    xmlns:t="http://manfred.jakesch.at/Beispiel/SOAPHeader"
    env:mustUnderstand="true">
      79
  </t:Transaktion>
</env:Header>
```

Listing 2: SOAP Header mit einem SOAP Header Block

Dieser Body stellt Mittel zur Verfügung, um Informationen an den ultimativen SOAP Empfänger zu liefern.
Das `Body` Element Information Item enthält folgende Einträge:

- einen lokalen Namen `Body`
- Namensraum mit „http://www.w3.org/2003/05/soap-envelope“
- beliebig viele namensraumqualifizierte Attribute
- beliebig viele namensraumqualifizierte Elemente

Es gilt für alle Kindelemente des SOAP `Body` Elements folgendes:

- Der Name des Elements sollte mit einem Namensraum versehen sein.
- Sie dürfen eine unbestimmte Anzahl an Zeichen enthalten.
- Laut Spezifikation ist es einem Kindelement des SOAP Body erlaubt, eine beliebige Anzahl von Kindelementen zu beinhalten.
- So ein Kindelement darf beliebig viele Attribute besitzen. Unter diesen Attributen darf auch das `encodingStyle` Attribut sein, welches eine besondere Bedeutung hat, die in einem vorhergehenden Abschnitt bereits näher erläutert wurde.

In der SOAP Spezifikation ist mit dem SOAP Fault ein einziges Kindelement des SOAP `Body` Elements vordefiniert worden. Dieses Element wird dazu benutzt, um eine geordnete Fehlerbehandlung zu ermöglichen. Dem SOAP Fault ist der folgende Abschnitt gewidmet.

2.4.4 Der SOAP Fault

Ein sogenannter SOAP Fault wird dazu benutzt, um Fehlerinformation mittels einer SOAP Nachricht zu transportieren.
Das `Fault` Element besteht aus folgenden Teilen:

- ein lokaler Name `Fault`
- ein Namensraum „http://www.w3.org/2003/05/soap-envelope“
- zwei oder mehr Kindelemente in folgender Ordnung:
  1. ein verpflichtendes `Code` Element: Dieses Element enthält ein verpflichtendes `Value` Element, das einen maschinell verarbeitbaren Fehlercode enthält. Es kann auch optionale `Subcode` Elemente enthalten. Näheres zu SOAP Fault Codes wird im Abschnitt 2.4.5 erläutert.
  2. ein verpflichtendes `Reason` Element: Dieses Element dient dazu um den aufgetretenen Fehler in menschlich lesbarer Form zu beschreiben. Es enthält mindestens ein `Text` Element, das die Fehlermeldung in mehreren Sprachen enthalten kann. Die verwendete Sprache wird über das Attribut `xml:lang` angegeben.
  3. ein optionales `Node` Element: Es enthält Informationen über den Knoten, der den Fehler verursachte. Dazu enthält das Element einen Wert vom Typ *xsd:anyURI*, der den fehlerhaften Knoten adressieren soll.
  4. ein optionales `Role` Element: Dieses Element enthält die Rolle, in der der SOAP Knoten agierte, als der Fehler auftrat.
  5. ein optionales `Detail` Element: Dieses Element dient zum Transport applikationsspezifischer Fehlerinformation, die in Beziehung zum SOAP Body steht. Es kann dazu benutzt werden, um den SOAP Fault Code näher zu beschreiben.

```
<env:Envelope xmlns:env="http://www.w3.org/2003/05/soap-envelope"
              xmlns:m="http://manfred.jakesch.at/Beispiel/Timeouts"
              xmlns:xml="http://www.w3.org/XML/1998/namspace"
  <env:Body>
    <env:Fault>
      <env:Code>
        <env:Value>env:Sender</env:Value>
        <env:Subcode>
          <env:Value>m:MessageTimeout</env:value>
```

```
      </env:Subcode>
    </env:Code>
    <env:Reason>
      <env:Text xml:lang="de">Sender empfing Timeout</env:Text>
      <env:Text xml:lang="en">Sender got a timeout</env:Text>
    </env:Reason>
    <env:Detail>
      <m:MaxTime>P6M</m:MaxTime>
    </env:Detail>
  </env:Fault>
 </env:Body>
</env:Envelope>
```

Listing 3: SOAP Fault Beispiel mit einem `Detail` Element das im Kontext des „env:Sender“ und „m:MessageTimeout“ Fault Codes interpretiert wird

Um als SOAP Fehlermeldung erkannt zu werden, ist es vorgeschrieben, dass ein SOAP *Body* Element nur ein Kindelement besitzt, das ein SOAP Fault Element ist. Daher darf ein SOAP Sender, der einen SOAP Fault generiert, keine zusätzlichen Elemente in den SOAP Body packen. Eine Nachricht mit einem SOAP Fault Element und zusätzlichen Elementen hat keine in SOAP definierte Semantik. SOAP Fault Elemente dürfen ebenso wenig in SOAP Headern wie auch in Kindelementen des SOAP *Body* Elements vorkommen. In Listing 3 wird eine SOAP Nachricht mit einem darin enthaltenen SOAP Fault dargestellt. Das `Code` Element des Faults enthält einen definierten Faultcode als Wert und ein `Subcode` Element, das Informationen darüber enthält, wie die Daten im `Detail` Element genau interpretiert werden sollten. Das `Detail` Element liefert eine genauere Spezifizierung des Fehlers der in dem `Subcode` Element aufgeführt wurde.

2.4.5 Die SOAP Fault Codes

SOAP Fault Codes sind XML-erweiterte Namen, die eine Klassifizierung von Fehlern zulassen. Ein SOAP Fault enthält eine hierarchische Liste von `Code` und `Subcode` Elementen, die den aufgetretenen Fehler mit jeder Stufe detaillierter beschreiben.
In einem `Value` Element eines SOAP `Code` Elements dürfen nur die in Tabelle 2 angeführten Werte vorkommen. Es können aber applikationsspezifische `Subcode` Elemente angelegt werden, um den Fehler genauer zu beschreiben. Die SOAP Version 1.2 Spezifikation definiert kein Limit für die Anzahl der `Subcode` Elemente, die ein SOAP Fault enthalten darf. Es wird aber angemerkt, dass eine relativ kleine Anzahl an `Subcode` Element ausreichen sollte.

lokaler Name	Bedeutung
VersionMismatch	Der den Fehler produzierende Knoten fand ein ungültiges Element Information Item statt des erwarteten `Envelope` Elements. Das Namensraumattribut, der lokale Name oder beides passen nicht zur Spezifikation des `Envelope` Elements in [39]. Wird ein Fehler mit diesem Code generiert, so sollte der generierende Knoten einen SOAP Header Block `Upgrade` einfügen, der eine Liste der unterstützten SOAP Envelopes sprich SOAP Versionen enthält.
MustUnderstand	Ein unmittelbares Kindelement des `Header` Elements konnte am empfangenden Knoten nicht verstanden werden, obwohl der Wert des `mustUnderstand` Attributs auf „true“ gesetzt war. Ein Knoten, der diesen Fehler generiert, sollte in der SOAP Message die den SOAP Fault enthält, einen Header Block einfügen, der ein `notUnderstood` Element enthält, das angibt, welcher Header Block nicht verarbeitet werden konnte.
DataEncodingUnknown	Ein SOAP Header Block oder ein SOAP Body Kindelement enthält eine Datenkodierung, die der empfangende Knoten nicht unterstützt. Die Daten konnten nicht dekodiert werden.
Sender	Die Nachricht war ungültig formatiert oder enthielt nicht die gewünschte Information, die notwendig gewesen wäre, um die Nachricht ordnungsgemäß zu verarbeiten. Als Beispiel wäre hier anzuführen, dass die Nachricht keine eventuellen Transaktionsdaten oder keine passenden Authentifizierungsdaten enthielt. Dieser Fehlercode gibt an, dass das erneute Senden der Nachricht ohne eine Änderung mit größter Wahrscheinlichkeit nicht zielführend ist.
Receiver	Die empfangene Nachricht konnte nicht verarbeitet werden. Dies hat aber nichts mit dem Inhalt der Nachricht zu tun, sondern ist eher darauf zurückzuführen, dass es andere Probleme bei der Abarbeitung der Nachricht gab. Dies könnte zum Beispiel beinhalten, dass ein Knoten am Weg der Nachricht ausgefallen ist, und deshalb die Nachricht unzustellbar war. Die Nachricht könnte also erfolgreich bearbeitet werden, wenn sie zu einem späteren Zeitpunkt noch einmal verschickt würde.

Tabelle 2: SOAP Fault Codes definiert in der Spezifikation [39]

2.5 Das SOAP Verarbeitungsmodell

Beim SOAP Processing Model handelt es sich um ein verteiltes Verarbeitungsmodell. Es wird davon ausgegangen, dass eine SOAP Nachricht von einem Sender abgeschickt wird und auf seinem Weg zum Empfänger beliebig viele SOAP Intermediaries passiert. Dieses SOAP Processing Modell kann mehrere Message Exchange Pattern (MEP) unterstützen. Als Beispiel seien hier nur Einweg-Nachrichten (one-way messages), Request/Response Interaktionen und Peer-to-Peer Konversationen genannt.

2.5.1 Verarbeitung von SOAP Nachrichten

Die SOAP Spezifikation legt eindeutig fest, in welcher Reihenfolge eine empfangene SOAP Nachricht verarbeitet werden muss. Es ist die angegebene Reihenfolge der durchzuführenden Schritte zu beachten.
Im Speziellen sind folgende Verarbeitungsschritte durchzuführen:

1. Es ist festzustellen, in welchen Rollen der Knoten, der soeben die Nachricht empfangen hat, agieren soll. Es ist dem Knoten erlaubt, in der SOAP Nachricht den SOAP Envelope, etwaige SOAP Header Blöcke und den SOAP Body nach möglichen Rollen des Knotens zu untersuchen.

2. Der Knoten muss alle für ihn verpflichtend zu verarbeitenden Header Blöcke identifizieren.

3. Werden ein oder mehrere SOAP Header Blöcke identifiziert, die aber nicht verstanden werden konnten, so muss ein SOAP Fault generiert werden, dessen `Code` Element den Wert „env:MustUnderstand“ enthält. Wurde so ein Fehler generiert, so ist die Abarbeitung der SOAP Nachricht zu stoppen. Fehler, die mit dem Inhalt des SOAP Body zu tun haben, dürfen danach nicht mehr generiert werden.

4. Es sind alle als verpflichtend identifizierten Header Blöcke zu bearbeiten. Handelt es sich bei dem empfangenden Knoten um einen ultimativen Empfänger, so ist auch der SOAP Body der Nachricht zu verarbeiten. Ein SOAP Knoten darf auch SOAP Header Blöcke bearbeiten, die optional sind.

5. Handelt es sich bei dem empfangenden SOAP Knoten um einen SOAP Intermediary und konnte die Nachricht ohne Fehler verarbeitet werden, so soll die Nachricht weitergeleitet werden.

In allen Fälle, in denen ein Header Block verarbeitet wird, muss der SOAP Knoten den SOAP Header auch verstehen und die Verarbeitung konform mit der Spezifikation des Header Blocks durchführen. Die erfolgreiche Abarbeitung eines SOAP Header Blocks garantiert keine erfolgreiche Abarbeitung eines Blocks des selben Namens in der gleichen SOAP Nachricht. Die Spezifikation des SOAP Header Blocks definiert die Umstände, in denen ein SOAP Fault generiert werden sollte.
Ein ultimativer Empfänger muss den SOAP Body eines Nachricht verarbeiten. Es müssen alle unmittelbaren Kinder des SOAP `Body` Elements korrekt berücksichtigt werden. Mit der Ausnahme des SOAP Fault Elements definiert die SOAP Spezifikation keine bestimmte Struktur für diese Elemente.
Fehler bei der Abarbeitung einer Nachricht werden mit einem SOAP Fault quittiert. Das Abarbeiten einer SOAP Nachricht darf aber nur zu höchstens einem SOAP Fault führen.
Bei der Abarbeitung einer SOAP Nachricht kann es natürlich vorkommen, dass mehrere Fehler entdeckt werden. Außer bei den im oberen Punkt identifizierten SOAP Header Fehlern ist es dem SOAP Knoten freigestellt, welcher Fehler übermittelt wird. Es bleibt aber sicherzustellen, dass nur ein SOAP Fault generiert wird.
Die Abarbeitung von einem oder mehreren SOAP Header Blöcken kann die Abarbeitungsreihenfolge von anderen SOAP Header Blöcken und/oder dem SOAP Body bestimmen. Sind solche bestimmenden Header Blöcke in der Nachricht nicht vorhanden, so ist es dem SOAP Knoten freigestellt, in welcher Reihenfolge die SOAP Header Blöcke und der SOAP Body abgearbeitet werden.

2.6 Die Sicherheit von SOAP

Das SOAP Messaging Framework definiert an sich keine Möglichkeiten, um Zugriffskontrollen, Vertraulichkeit und Integrität von Nachrichten sicherzustellen. Diese Mechanismen können aber als Erweiterungen im Sinne des SOAP Extensibility Modells zur Verfügung gestellt werden.
Für SOAP Implementierungen ist es wichtig, dass sie Möglichkeiten vorsehen, um SOAP Knoten zu identifizieren, die bösartige Daten verschicken. Es wird in der SOAP Spezifikation darauf hingewiesen, dass es SOAP Knoten möglich sein sollte, den Grad an Vertrauen, den sie einem anderen Knoten gegenüber aufbringen sollen, selbst zu bestimmen.

2.6.1 Sicherheitsüberlegungen bezüglich SOAP Knoten

SOAP transportiert applikationsdefinierte Daten als SOAP Header Blöcke oder SOAP Body Inhalt. Das Abarbeiten von SOAP Header Blöcken kann zu Seiteneffekten wie Zustandsänderungen, Archivieren von Informationen oder der Generierung von weiteren Nachrichten führen. Deshalb ist bei Header Blöcken besonders darauf zu achten, dass nur den Spezifikationen entsprechende Header Blöcke abgearbeitet werden, und man sich den Sicherheitsrisiken der jeweiligen Seiteneffekte bewusst ist.
Ebenso gilt bei der Abarbeitung des Inhalts eines SOAP `Body` Elements, dass Seiteneffekte nur in einem gewissen Sicherheitsrahmen auszuführen sind. Es dürfen nur wohlgeformte SOAP Body Inhalte bearbeitet werden.
Wird die SOAP Nachricht mittels modularer Software verarbeitet, so ist darauf zu achten, dass jedes Modul Zugriff auf den gesamten Sicherheitskontext hat. Als Beispiel wäre hier anzuführen, dass ein SOAP Body nicht verarbeitet werden sollte, ohne über den Kontext des Empfangs Details zu wissen.

2.7 Das SOAP Datenmodell

Das SOAP Datenmodell in [40] stellt applikationsspezifische Datenstrukturen und Werte als einen gerichteten, beschrifteten Graphen dar.
Es ist mit dem SOAP Datenmodell nicht vorgesehen, dass man XML-basierte Daten in eine übertragungsfähige Form bringt. Es ist in der Spezifikation angemerkt, dass das in [40] dargestellte Datenmodell, das dazugehörige SOAP Encoding Schema und die SOAP RPC Repräsentation optional sind. Applikationen, die ihre Daten bereits in XML darstellen, brauchen das SOAP Datenmodell nicht zu benutzen.
Da die Nichtunterstützung des SOAP Datenmodells aber die Interoperabilität zwischen den SOAP Implementierungen stark beeinflusst beziehungsweise teilweise unmöglich macht, wird im folgenden Abschnitt auf dieses Datenmodell näher eingegangen.

2.7.1 Die Kanten des Graphen

Bei den Kanten eines Graphen wird zwischen eingehenden und ausgehenden Kanten unterschieden. Eine Kante kann sowohl ausgehende als auch eingehende Kante des selben Knotens sein. Ebenso ist es möglich, dass eine Kante nur ausgehend beziehungsweise nur eingehend ist.
Die ausgehenden Kanten eines Knotens können durch sogenannte Labels oder durch die Position unterschieden werden. Die Position ist die totale Ordnung aller solcher Kanten. Wird also eine ausgehende Kante durch ihre Position

unterschieden, so gilt dieses Kriterium für sämtliche ausgehende Kanten eines Knotens.
Ein Label ist ein XML-qualifizierter Name. Zwei Kantenlabels sind nur dann gleich, wenn deren XML-erweiterte Namen gleich sind. Dies bedeutet, dass die zwei nachfolgenden Bedingungen erfüllt sind:

1. Der Wert des lokalen Namens ist gleich.
2. Eine der beiden folgenden Aussagen ist wahr:
   (a) Bei beiden Labels fehlt der Namensraum.
   (b) Die beiden Namensräume sind vorhanden und äquivalent.

Für nähere Informationen bezüglich dem Vergleich von XML-qualifizierten Namen ist die XML Schema Spezifikation in [12] zu konsultieren.

2.7.2 Die Knoten des Graphen

Ein Knoten hat beliebig viele ausgehende Kanten. Ein Graph, der keine ausgehenden Kanten hat, hat einen optionalen lexikalischen Wert. Alle Knoten eines Graphen haben einen optionalen Typnamen vom Typ *xsd:QName* im Namensraum „http://www.w3.org/2001/XMLSchema“ ([12]).
Ein Knoten kann single reference oder multi reference sein. Ein single reference Graphknoten hat eine einzige eingehende Kante. Ein multi reference Knoten hat mehrere eingehende Kanten.

2.7.3 Werte

Ein einfacher Wert (simple value) ist ein Graphknoten mit einem lexikalischen Wert.
Ein zusammengesetzter Wert (compound value) wird durch einen Knoten mit beliebig vielen ausgehenden Kanten wie folgt definiert:

1. Ein Graphknoten, dessen ausgehende Kanten nur durch die Labels der Kanten unterschieden werden können, wird als Struktur (struct) bezeichnet. Die ausgehenden Kanten müssen also mit eindeutigen Labels versehen werden.
2. Ein Graphknoten, dessen ausgehende Kanten sich nur durch die Position unterscheiden, wird Feld (array) genannt. Diese ausgehenden Kanten dürfen keine Labels tragen.

2.8 SOAP Encoding

SOAP Encoding bietet die Möglichkeit, Daten, die dem SOAP Datenmodell entsprechen, zu kodieren. Es ist nicht relevant, ob es sich im SOAP Header oder SOAP Body Daten handelt. Ebenso können aber auch unkodierte Daten oder andere Datenmodelle in SOAP Nachrichten verwendet werden.
Die Serialisierungsregeln, die in diesem Kapitel vorgestellt werden, sind mit der URI „http://www.w3.org/2003/05/soap-encoding“ identifiziert. SOAP Nachrichten, die dieses Encoding Schema unterstützen, sollten mittels dem Attribut `encodingStyle` (siehe Abschnitt 2.4.1) dies zur Kenntnis bringen.

2.8.1 Zuweisungen zwischen XML und dem SOAP Datenmodell

XML erlaubt eine flexible Kodierung von Daten. SOAP Encoding stellt eine Menge von Regeln zur Verfügung, wie Knoten der vorher beschriebenen Graphen kodiert werden können.
Typischerweise gibt es mehrere Möglichkeiten, um einen Graphen zu kodieren. Wird ein Graph serialisiert, so ist darauf zu achten, dass die gewählte Serialisierungsform dann auch dem abzubildenden Graphen entspricht. Gibt es mehrere solche Darstellungsmöglichkeiten, so kann man eine dieser Möglichkeiten auswählen. Empfängt man eine kodierte SOAP Nachricht, so müssen alle möglichen Ausformungen eines Graphen akzeptiert werden.

2.8.1.1 Kodierung von Knoten und Kanten

Jede Kante wird als Element Information Item kodiert und jedes Element Information Item wird als Kante des Graphen dargestellt.
Der Knoten eines Graphen auf den eine Kante zeigt, wird wie folgt bestimmt:

- Enthält das Element, das die Kante repräsentiert, kein `ref` Attribut, dann stellt dieses Element einen Knoten dar, und die Kante zeigt auf diesen Knoten. In solchen Fällen repräsentiert ein Element Information Item sowohl eine Kante als auch einen Knoten.

- Enthält das Element ein `ref` Attribut, dann muss der Wert dieses Attributs genau einem `id` Attribut in dem SOAP Envelope entsprechen. In diesem Fall zeigt die Kante auf den Knoten, der das Element mit dem entsprechenden `id` Attribut enthält. Dieses Element Information Item muss im Scope eines `encodingStyle` Attributs mit dem Wert „http://www.w3.org/2003/05/soap-encoding“ sein.

2.8.1.2 Kodierung von einfachen Werten

Ein lexikalischer Wert eines Knotens stellt einen einfachen Wert dar. Dieser ist eine Folge von Unicode Zeichen. Das Element Information Item, welches einen einfachen Wert darstellt, kann ein Attribut namens `nodeType` enthalten. Der Wert dieses Attributs muss dann `simple` sein.

2.8.1.3 Kodierung von zusammengesetzten Werten

Die ausgehende Kante eines Knotens wird als Kindelement des Elements dargestellt, das dem Knoten entspricht. Je nach Typ des zusammengesetzten Wertes, den der Knoten darstellt, werden bestimmte Regeln angewendet. Diese sind folgende:

1. Für eine Kante, welche durch Labels unterschieden wird, bestimmen der lokale Name und der Namensraum des Kindelements den Wert des Labels.

2. Für eine Kante, die durch ihre Position unterschieden wird, gilt folgendes:

   - Die ordinale Position der Kante entspricht der Position des Kindelements unter seinen Geschwistern.
   - Der lokale Name und der Namensraum sind nicht signifikant.

3. Ein Element, das einen zusammengesetzten Wert darstellt, kann ein `nodeType` Attribut enthalten. Dieses hätte dann den Wert `struct` oder `array`.

4. Die folgenden Regeln gelten für einen Knoten, der ein `array` darstellt:

   - Das Element einer SOAP Nachricht, welches einen `array` Knoten darstellt, kann ein `itemType` Attribut enthalten. Dieses Attribut wird dazu verwendet, um den Typnamen eines Elements zu berechnen.
   - Das Element kann ein `arraySize` Attribut enthalten. Dieses Attribut gibt die Größe und Dimension des Feldes an. Der Standardwert für dieses Attribut ist ein eindimensionales Feld mit nicht spezifizierter Größe.

5. Zeigt eine Kante nicht auf einen Knoten, so kann es nicht serialisiert werden, oder das entsprechende Element bekommt ein Attribut *xsd:nil* mit dem Wert `true`.

2.8.2 Dekodierungsfehler

Folgende Fehler sollte ein SOAP Empfänger melden, wenn Probleme mit der Dekodierung der Daten einer SOAP Nachricht im Spiel waren:

- Der SOAP Receiver sollte einen „env:Sender“ SOAP Fault mit einem Subcode `enc:MissingID` liefern, wenn die Nachricht ein `ref` Attribut enthält, aber kein passendes `id` Attribut gefunden werden konnte, also eine Referenz nicht existiert.
- Der SOAP Empfänger sollte einen „env:Sender“ SOAP Fault mit Subcode `enc:DuplicateID` generieren, wenn es Elemente gibt, die ein `id` Attribut mit dem selben Wert enthalten.
- Der SOAP Receiver könnte einen „env:Sender“ SOAP Fault mit Subcode `enc:UntypedValue`, wenn der Typname für einen kodierten Knoten des Graphen nicht spezifiziert wurde, die zugrunde liegende SOAP Implementierung eine Typisierung des Knotens aber verlangt.

2.9 SOAP RPC Darstellung

In der Spezifikation zu SOAP Version 1.2 wurde speziell ein Kapitel dem Thema der Durchführung von Remote Procedure Calls mittels SOAP gewidmet. Dies schließt aber die Verwendung von anderen Message Exchange Pattern in Verbindung mit SOAP nicht aus.
Da es sich bei SOAP RPC um den im Moment häufigsten Anwendungsfall von SOAP Nachrichten handelt, soll dieser Abschnitt Einblicke in die Umsetzung geben.
Das SOAP RPC Modell ist an kein spezielles Transportprotokoll gebunden, jedoch ist eine Umsetzung auf HTTP auf Grund der Charakteristik dieses Protokolls nur natürlich.
Um einen RPC durchzuführen, sind folgende Daten von Nöten ([19]):

- die Adresse des SOAP Knoten, der die Prozedur ausführen soll. Beim HTTP Binding etwa wäre das eine URL (Uniform Resource Locator).
- der Prozedur- oder Methodenname der Service-Funktion, die ausgeführt werden soll.
- Identitäten und Argumente, die von der Prozedur oder Methode verlangt werden zusammen mit Rückgabewerten und Ausgabeparametern.

- eine klare Unterscheidung der Argumente, die notwendig sind, um die Webressource eindeutig zu identifizieren und der Nutzdaten und Kontrollinformationen, die zur Abarbeitung des Aufrufs notwendig sind.
- optionale SOAP Header

2.9.1 Umsetzung von SOAP RPC im SOAP Body

Grundsätzlich werden bei SOAP RPCs Aufruf und Antwort im SOAP `Body` Element kodiert und in diesem Element auch versandt.
Ein Aufruf eines RPC wird wie folgt im Graphenmodell dargestellt:

- Der Aufruf ist eine einzige Struktur („struct“), die eine ausgehende Kante für jeden Eingabe- beziehungsweise Ein-Ausgabeparameter enthält. Diese Struktur bekommt den Namen der Funktion oder Methode, die aufgerufen wurde.
- Jede ausgehende Kante hat ein Label, das den Namen des Parameters, den die Kante symbolisiert, trägt.

```
<env:Envelope xmlns:env="http://www.w3.org/2003/05/soap-envelope"
  <env:Body>
    <m:bestaetigeReservierung
      env:encodingStyle="http://www.w3.org/2003/05/soap-encoding/"
      xmlns:m="http://manfred.jakesch.at/Beispiel">
      <m:reservierungscode>
        1234gth345
      </m:reservierungscode>
    </m:bestaetigeReservierung>
  </env:Body>
</env:Envelope>
```

Listing 4: SOAP RPC Request Beispiel

```
<env:Envelope xmlns:env="http://www.w3.org/2003/05/soap-envelope"
  <env:Body>
    <m:bestaetigeReservierungResponse
      env:encodingStyle="http://www.w3.org/2003/05/soap-encoding/"
      xmlns:m="http://manfred.jakesch.at/Beispiel">
      <rpc:result>
        m:status
      </rpc:result>
      <m:status>
        Transaktion war erfolgreich.
```

```
      </m:status>
      <m:bestaetigungscode>
        12f45t4rzrz
      </m:bestaetigungscode>
    </m:bestaetigeReservierungResponse>
  </env:Body>
</env:Envelope>
```

Listing 5: SOAP RPC Response Beispiel

Listing 4 zeigt einen Funktionsaufruf der Methode „bestaetigeReservierung“, die als Eingabeparameter einen „reservierungscode“ verlangt. Die Antwort auf diesen RPC Request zeigt Listing 5. Die Antwort enthält einen Ausgabeparameter und einen Rückgabewert, der einen Verweis auf ein Element in der Ergebnisstruktur hat. Die Struktur, die die Antwort des RPC enthält, erhält per Konvention den Namen der ausgeführten Funktion mit dem Postfix „Response“.
Laut der Spezifikation von SOAP Version 1.2 in [40] ist es den Applikationen freigestellt, ob sie Funktionsaufrufe mit unvollständigen Parameterangaben ausführen oder nicht. Einen Vergleich der Möglichkeiten bei den untersuchten SOAP Kommunikationsplattformen findet sich in Abschnitt 4.2.2.7.

2.10 Transportprotokolle für SOAP

SOAP verlässt sich beim Transport der Nachrichten auf sogenannte darunter liegende Protokolle (engl. underlying protocols). Diese Protokolle stellen eine gewisse Bereicherung der SOAP Funktionen dar, was Verlässlichkeit, Verbindungssicherheit und Sicherheit betrifft.
Die Spezifikation, wie eine SOAP Nachricht von einem Knoten zum anderen mittels eines darunter liegenden Protokolls übertragen wird, nennt man SOAP Binding. Die SOAP Spezifikation in [39] beschreibt eine SOAP Nachricht als ein sogenanntes XML Infoset, das aus einem abstrakten Dokument, dem SOAP Envelope, besteht. Ein SOAP Protokoll Binding ist unter anderem dafür zuständig, dass ein solches Infoset an einem Knoten serialisiert wird, zum anderen übertragen wird und am anderen Knoten ein identisches Infoset rekonstruiert werden kann. Ein typisches Beispiel für so eine Serialisierung eines XML Infosets zeigen die beispielhaften SOAP Nachrichten dieser Arbeit. Diese bestehen nämlich alle aus wohlgeformten XML Dokumenten.
Neben der Möglichkeit der Übertragung von SOAP Infosets unterstützt ein SOAP Protokoll Binding auch sogenannte SOAP Features. Diese Features sind gewisse Funktionalitäten, die vom Binding übernommen werden können. Beispiele für solche Features wären eine Korrelation zwischen Aufruf und

Antwort herzustellen, einen verschlüsselten Kanal bereitzustellen oder eine verlässliche Zustellung zu garantieren. Die Features eines SOAP Bindings müssen in der Spezifikation dieses Bindings angeführt werden. Es bleibt aber anzumerken, dass nicht alle Protokolle die selben Features zur Verfügung stellen können. Wird also ein Feature nicht durch das SOAP Binding definiert, so gibt es noch die Möglichkeit, dass ein solches Feature in der SOAP Nachricht implementiert wird. Dies geschieht durch Hinzufügen von etwaigen SOAP Header Elementen. Ist ein Feature mittels SOAP Header Blöcken spezifiziert, so spricht man von einem SOAP Modul.
Das am meisten verwendete Protokoll, das auch bei allen Programmiersprachen hauptsächlich unterstützt wird, ist HTTP (Hypertext Transfer Protocol). Aus diesem Grund wird im folgenden Kapitel genauer betrachtet, was die SOAP Spezifikation in [40] für eventuelle Anwendungsfälle normiert hat.
Dies heißt aber nicht, dass HTTP das einzige Transportprotokoll für SOAP ist. So gibt es in [27] des W3C ein exemplarisches Protokoll Binding von SOAP und SMTP zu finden. Eine Erweiterung auf ein beliebiges Transportprotokoll ist also denkbar und teilweise schon durchdacht.

2.10.1 Das SOAP HTTP Binding

HTTP hat ein bekanntes Verbindungsmodell und Message Exchange Pattern. Der Client identifiziert einen Server mittels einer URI. Für Verbindungen wird das TCP/IP Netzwerk verwendet. Vom Client wird ein HTTP Request an den Server geschickt, der wiederum mit einer HTTP Response antwortet. Implizit wird also hier eine Verbindung zwischen Aufruf und Antwort hergestellt. Stellt man eine Verbindung zu SOAP her, so wird man bemerken, dass man diese Art der Verbindung auf bequeme Art und Weise zur Erstellung eines SOAP RPC nutzen kann. Ein SOAP Knoten wird dann mittels der Request-URI identifiziert.
Das HTTP Binding in [40] benützt das sogenannte SOAP Web Method Feature, welches eine Kompatibilität der SOAP Anwendungen mit den Prinzipien des World Wide Web sicherstellen soll. Dieses Feature wird bei SOAP in Verbindung mit HTTP auf die „generischen“ Methoden GET und POST eingeschränkt. Für nähere Informationen zu den Webmethoden in Bezug auf HTTP siehe [34].
Ein HTTP GET würde also dem SOAP Message Exchange Pattern „SOAP Response Message Exchange Pattern“ entsprechen. Es geht darum, dass ein Nicht-SOAP-Befehl an einen Knoten gesendet wird und dann von einem SOAP Knoten mit einer SOAP Nachricht beantwortet wird. Dieser Nachrichtenaustausch hat keine Auswirkungen auf die Daten, die der SOAP Knoten eventuell verwaltet oder verarbeitet. Hier handelt es sich lediglich um eine

Statusabfrage. Solche Anfragen werden als idempotent und sicher bezeichnet, da sie nur zur Gewinnung von Information dienen. Dies würde in etwa dem Abfragen einer Webseite mit dem Browser entsprechen.
Listing 6 zeigt beispielhaft eine Abfrage der Reservierungsdaten mittels eines HTTP GET. Interessant ist hier auch der HTTP Header Accept. Dieser deutet mit dem Wert „application/soap+xml“ an, dass die HTTP Response als SOAP Nachricht formatiert werden soll, damit sie maschinell verarbeitbar ist.
Ein HTTP POST entspricht einem „SOAP Request-Response Message Exchange Pattern“. Beim HTTP Request wird eine SOAP Nachricht an den durch die URI identifizierten Knoten geschickt, und der Knoten antwortet in der HTTP Response mit einer SOAP Nachricht auf den Aufruf. Dieses Message Exchange Pattern kann von allen Applikationen verwendet werden, egal ob sie nun RPC verwenden oder über diesen Weg einfach XML Daten in SOAP Nachrichten verschicken. HTTP POST hat den Vorteil, dass auch SOAP Header mit dem HTTP Request verschickt werden können. Es gilt zu beachten, dass der HTTP Content Type Header den Wert „application/-soap+xml“ enthalten muss, damit eine korrekte Identifizierung der SOAP Nachricht gewährleistet werden kann.
SOAP Nachrichten, die mittels HTTP verschickt werden, folgen der Semantik der HTTP Statuscodes für die Übermittlung von Statusinformationen in HTTP. Zum Beispiel zeigt ein Statuscode von 200 an, dass die Nachricht erfolgreich verarbeitet werden konnte. Eine Aufstellung der am häufigsten verwendeten Statuscodes und deren Bedeutung in Bezug auf SOAP findet sich in Tabelle 3.
In Listing 7 wird das selbe Beispielservice aufgerufen wie in Listing 6. Nur wird beim erstgenannten ein HTTP POST verwendet, um den notwendigen Parameter für die Funktion zu übergeben. Beim HTTP GET Beispiel in Listing 6 wurde schon in der URI, die das Service identifiziert, der „code“ Parameter angegeben. Wie bereits weiter oben erwähnt, ist es ein Vorteil von HTTP Post, dass man in der SOAP Anfrage auch SOAP Header unterbringen kann, was mittels eines HTTP Get Requests nicht möglich ist.

2.10.2 Alternative SOAP Bindings

In diesem Abschnitt sollen kurz alternative SOAP Bindings besprochen werden. Aufgrund der Konzeption von SOAP sind viele mögliche Varianten von SOAP Bindings denkbar. Alleine das SOAP Binding für HTTP hat es aber in die Recommendation des W3C geschafft.
Grundsätzlich gibt es in vielen SOAP Implementierungen auch schon alternative SOAP Bindings, die zusätzlich zu HTTP verwendet werden können.

```
GET /manfred.jakesch.at/Beispiel/Reservierung?code=12345 HTTP/1.1
Host: manfred.jakesch.at
Accept: application/soap+xml

HTTP/1.1 200 OK
Content-Type: application/soap+xml; charset="UTF-8"
Content-Length: 1234

<?xml version='1.0' ?>
<env:Envelope xmlns:env="http://www.w3.org/2003/05/soap-envelope"
  <env:Header>
    <t:Transaktion
      xmlns:t="http://manfred.jakesch.at/Beispiel/Transaktion"
      env:mustUnderstand="true">
        79
    </t:Transaktion>
  </env:Header>
  <env:Body>
    <m:Reservierung
      env:encodingStyle="http://www.w3.org/2003/05/soap-encoding/"
      xmlns:m="http://manfred.jakesch.at/Beispiel">
      <m:Zimmer>
        203B
      </m:Zimmer>
      <m:Auto>
        Mercedes SLK
      </m:Auto>
    </m:Reservierung>
  </env:Body>
</env:Envelope>
```

Listing 6: Beispiel eines HTTP GET Requests und die Antwort eines SOAP Knotens

Status	**Begründung**	**Beschreibung**
2xx	Successful	Die Anfrage war erfolgreich.
200	OK	Die Anfrage war in Ordnung. Die Antwort folgt im HTTP Body.
3xx	Redirection	Der SOAP Knoten ist „umgezogen“. Dies bedeutet, dass der Aufruf mit der mitgesendeten URI noch einmal versucht werden sollte.
400	Bad Request	Dieser Statuscode deutet auf ein Problem mit der HTTP Request Nachricht hin. Er entspricht einem `env:Sender` SOAP Fault.
500	Internal Server Error	Dieser Statuscode zeigt ein Serverproblem oder ein Problem mit der empfangenen Nachricht an. Dieser Statuscode wird von den folgenden SOAP Faults benutzt: • `env:Sender` • `env:DataEncodingUnknown` • `env:Receiver` • `env:VersionMismatch` • `env:MustUnderstand`

Tabelle 3: HTTP Fehlercodes und ihre korrespondierenden SOAP Faults

```
POST /Beispiel/Reservierung HTTP/1.1
Host: manfred.jakesch.at
Content-Type: application/soap+xml; charset"UTF-8"
Content-Length: 1234

<?xml version='1.0' ?>
<env:Envelope xmlns:env="http://www.w3.org/2003/05/soap-envelope"
  <env:Header>
    ...
  </env:Header>
  <env:Body>
    <m:bestaetigeReservierung
      env:encodingStyle="http://www.w3.org/2003/05/soap-encoding/"
      xmlns:m="http://manfred.jakesch.at/Beispiel">
      <m:code>
        12345
      </m:code>
    </m:bestaetigeReservierung>
  </env:Body>
</env:Envelope>

HTTP/1.1 200 OK
Content-Type: application/soap+xml; charset="UTF-8"
Content-Length: 1234

<?xml version='1.0' ?>
<env:Envelope xmlns:env="http://www.w3.org/2003/05/soap-envelope"
  <env:Header>
    ...
  </env:Header>
  <env:Body>
    <m:Reservierung
      env:encodingStyle="http://www.w3.org/2003/05/soap-encoding/"
      xmlns:m="http://manfred.jakesch.at/Beispiel">
      ...
    </m:Reservierung>
  </env:Body>
</env:Envelope>
```

Listing 7: Beispiel eines HTTP POST Requests und die Antwort eines SOAP Knotens

Es gilt hier aber die Einschränkung, dass es mit ziemlicher Sicherheit zu Interoperabilitätsproblemen kommt, da diese Alternativen oft nur auf eine Plattform beschränkt sind. Problematisch ist hier eben der Umstand, dass außer dem HTTP Binding noch keine fertige Spezifikation für andere SOAP Bindings vorliegt.
Als eine Alternative könnte sich früher oder später das SMTP Binding von SOAP herausstellen. Im Dokument [27] wird exemplarisch ein SOAP Binding an das E-Mail Transportprotokoll beschrieben. Diese Beschreibung hat aber keinen normativen Charakter, sondern soll eher die Flexibilität und Erweiterbarkeit des SOAP Protokolls herausstreichen. Es findet aber schon in einigen ausgesuchten SOAP Implementierungen seine Umsetzung.
Weiters gibt es Versuche, dass man die SOAP Kommunikationsplattformen einfach über TCP Sockets miteinander kommunizieren lässt. Diese Bestrebungen sind aber bestenfalls als experimentell zu bezeichnen, da die meisten Implementierungen auf die Kommunikation mittels HTTP optimiert sind. Eine Ausnahme bildet die .NET Remoting Architektur von Microsoft. Zwischen zwei Parteien von .NET Anwendungen ist es problemlos möglich, mit SOAP Nachrichten über TCP Sockets zu kommunizieren.
Näheres zu diesem Thema beziehungsweise der Umsetzung von alternativen Transportprotokollen in den untersuchten SOAP Kommunikationsplattformen findet sich in Abschnitt 4.3.3.

2.11 Das SOAP Attachment Feature

Obwohl SOAP Version 1.2 als XML Infoset in [39] definiert wurde, kann man damit rechnen, dass der größte Teil der SOAP Daten als XML 1.0 (siehe [13]) dargestellt wird.
Es ergeben sich durch die obige Darstellung der SOAP Daten nach [53] folgende Probleme:

- Die Einbindung von binären Daten, wie zum Beispiel Bilder oder ähnliches, in die XML-Repräsentation kann nur durch einen Mehraufwand in Bezug auf Kodierung und Dekodierung der Binärdaten bewerkstelligt werden. Ein signifikanter Aufwand ist sowohl vom Transfervolumen für die kodierten Daten als auch für die Prozessorauslastung zu berücksichtigen.

- Es ist schwierig, andere XML Dokumente in eine bestehende SOAP Nachricht einzubetten. Besonders problematisch ist es, wenn das einzubettende XML-Dokument einen anderen Zeichensatz verwendet.

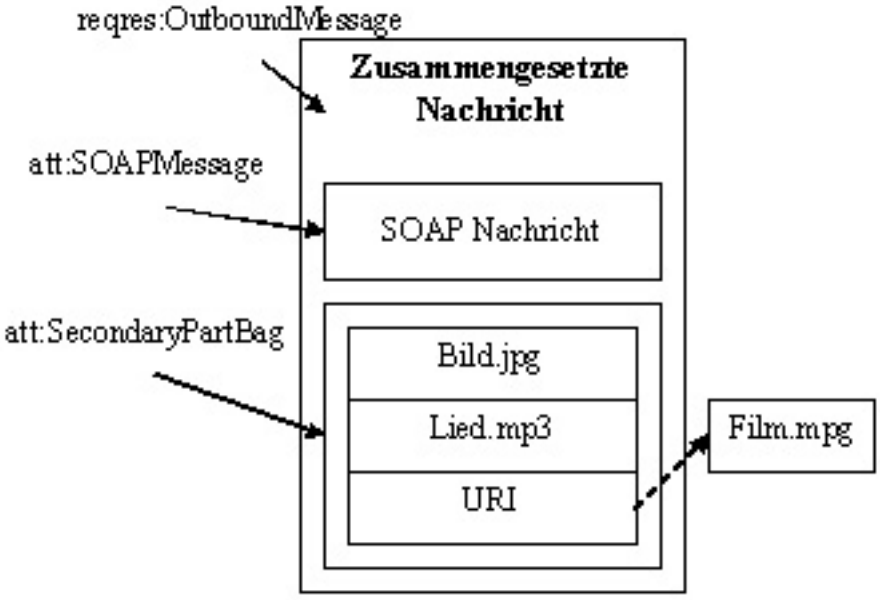

Abbildung 2: Aufbau einer zusammengesetzten Nachricht zur Übermittlung von Attachments nach [53]

- Um eine SOAP Nachricht nach Trennzeichen zu untersuchen, muss die gesamte Nachricht mittels eines Parsers bearbeitet werden. Dies stellt einen signifikanten Aufwand dar, will man etwa die eingebetteten Objekte aus der Nachricht extrahieren. Der gesamte Nachrichtenkontext muss berücksichtigt werden.

In [53] wird eine zusammengesetzte Nachrichtenstruktur (engl. „compound message structure“) definiert, die es ermöglicht, Binärdaten als sogenannte Attachments zu verschicken.
Abbildung 2 zeigt den schematischen Aufbau einer Nachricht. Diese besteht, in Abbildung 2 mit dem Namen `reqres:OutboundMessage` versehen, aus einer SOAP Nachricht, in der Abbildung `att:SOAPMessage` genannt, die Informationen zur Verarbeitung der Binärdaten enthält, und einem zweiten Teil, dem `att:SecondaryPartBag`, der aus keinem oder mehreren Attachments besteht. Dieser zweite Nachrichtenteil kann direkt Binärdaten enthalten oder diese mittels einer URI referenzieren.
Es bleibt festzuhalten, dass es sich bei der Spezifikation in [53] noch um einen sogenannten „Working Draft“ handelt, der noch keinen normativen Charakter hat. Grundsätzlich sieht man aber in dieser Spezifikation, die ein abstraktes Datenmodell vorstellt, wie eine Nachricht aufgebaut sein sollte, damit Nicht-

XML Daten zusammen mit einer SOAP Nachricht verschickt werden können. Für Implementierungen, die SOAP Attachments verschicken wollen, gelten folgende verbindliche Regeln:

- Der erste und zweite Teil einer zusammengesetzten Nachricht müssen dem Empfänger übermittelt werden können. Häufig wird dazu ein Kapselungsmechanismus benutzt, um einen einzelnen Datenstrom aus den verschiedenen Teilen zu erstellen. Dieser Datenstrom wird dann dem Empfänger übermittelt. Mögliche Mechanismen zur Kapselung von Daten bieten MIME (siehe [35]) oder DIME (siehe [52]).

- Es muss ein Mechanismus zur Verfügung stehen, der es erlaubt, die einzelnen Teile einer zusammengesetzten Nachricht mittels einer URI zu identifizieren.

Um konkret Binärdaten mittels SOAP zu verschicken, gibt es im Moment unter anderem die folgenden Möglichkeiten:

- Die SOAP Nachricht und die Binärdaten können in einer einzigen DIME Nachricht gekapselt werden und mittels TCP oder HTTP verschickt werden. Eine Beschreibung dieser Möglichkeit findet man in [52].

- Die Binärdaten werden zusammen mit der SOAP Nachricht in einer einzigen MIME Multipart/related Nachricht verschickt. Als Transportprotokoll würde sich hier HTTP anbieten. Diese Möglichkeit wurde schon für SOAP 1.1 in [10] ausführlich beschrieben.

- Es gibt aber auch noch die Möglichkeit, dass die SOAP Nachricht mittels HTTP übermittelt wird, die Binärdaten aber mittels separater HTTP GET Aufrufe geholt werden. In der SOAP Nachricht müssen also Referenzen auf Ressourcen im Netz angegeben werden.

Es sei aber hier noch erwähnt, dass die Möglichkeiten sich nicht auf die genannten beschränken.
Mittlerweile hat sich mit der SOAP Message Transmission Optimization Mechanism (SOAP MTOM) Spezifikation ein neuer Sachverhalt ergeben. Bei den vorher vorgestellten Formaten handelt es sich nicht mehr um reine SOAP Nachrichten, sondern es sind eben MIME- oder DIME-kodierte Nachrichten, die SOAP als Teil der Nachricht enthalten. Diese SOAP Nachrichten referenzieren ihrerseits andere Teile der kodierten Nachricht. Mit der SOAP MTOM Spezifikation soll sich das ändern. Für den Transport wird die SOAP Nachricht mit ihren Attachments kodiert, aber nach Ankunft am SOAP

Knoten wieder in eine echte SOAP Nachricht mit Binärteilen überführt. Es soll SOAP-basierten Applikationen erspart bleiben, mit den kodierten Nachrichten zu hantieren. Für eine genauere Beschreibung der Übermittlung von SOAP Attachments mittels SOAP MTOM siehe [41].
Im Moment gibt es aber noch keine Umsetzung dieser Spezifikation. Man darf aber gespannt sein, wie sich dieser neue Standard entwickelt. Microsoft hat bereits angekündigt, bald nur mehr diesen Standard zu unterstützen ([57]).
Näheres zur Umsetzung des SOAP Attachment Features in den untersuchten SOAP Kommunikationsplattformen findet sich in Abschnitt 4.3.2.

2.12 Unterschiede zwischen SOAP/1.1 und SOAP Version 1.2

Viele Implementierungen von SOAP unterstützen in erster Linie die Version 1.1. Es haben sich aber bei der Spezifikation der Version 1.2 einige Dinge geändert. Grund genug, um diese Unterschiede hier näher zu beleuchten.
Wie in [43] und [51] beschrieben, wird der Begriff SOAP nicht mehr als Akronym für „Simple Object Access Protocol“ verwendet, da diese Umschreibung etwas irreführend war. SOAP ist jetzt ein Eigenname.
Der augenscheinlichste Unterschied ist, dass es in Version 1.2 um XML Infosets geht und nicht mehr, wie in Version 1.1, um eine Serialisierung als XML 1.0 Dokument. Für eine nähere Beschreibung von XML Information Sets siehe [18].
Um die verschiedenen SOAP Versionen einfach auseinander halten zu können, damit die Implementierungen auch verschiedenste Versionen nebeneinander zur Verfügung stellen und bestehende Versionen weiter benutzt werden können, wurden unterschiedliche Namensräume für das Envelope Element und die Encoding Schemas für die beiden Versionen verwendet.

2.12.1 Änderungen am SOAP Envelope

Bei SOAP Version 1.2 ist nach dem SOAP Body Element kein weiteres Element mehr im SOAP Envelope einer SOAP Nachricht erlaubt. In SOAP/1.1 wird zu diesem Umstand geschwiegen, also ist es durchaus möglich, zusätzlich zum SOAP Body noch weitere XML Elemente als Kindelemente des SOAP Envelope anzuhängen. Tabelle 4 zeigt eine Gegenüberstellung der beiden Versionen bezüglich des Aufbaus eines SOAP Envelopes.
SOAP 1.2 erlaubt es nicht, dass das Attribut `encodingStyle` im SOAP Envelope Element vorkommt. In Version 1.1 aber darf dieses Attribut in jedem Element einer SOAP Nachricht vorkommen. In der neuesten Version dürfen also nur ausgesuchte Elemente dieses Attribut besitzen.

SOAP 1.1	SOAP 1.2
<env:Envelope ...> <env:Header> beliebig viele Header Blöcke </env:Header> <env:Body> SOAP Nachricht </env:Body> andere XML Elemente </env:Envelope>	<env:Envelope ...> <env:Header> beliebig viele Header Blöcke </env:Header> <env:Body> SOAP Nachricht </env:Body> </env:Envelope>

Tabelle 4: Gegenüberstellung des Aufbaus eines SOAP Envelopes

Bei Version 1.1 gibt es das SOAP Envelope Attribut `actor`. Dieses wurde in der neuesten Version in `role` umbenannt. Die Semantik dieses Attributs bleibt aber unverändert. Zu der bereits in der Vorgängerversion bekannten Rolle „next“ sind jetzt die Rollen „none“ und „ultimateReceiver“ dazugekommen. Mehr zu den Rollen in SOAP 1.2 findet man in Abschnitt 2.3 dieser Arbeit.
Die SOAP Fault Codes wurden auch überarbeitet. Es gibt jetzt keine „Client“ und „Server“ Faults mehr, sondern „Sender“ und „Receiver“ Faults. Außerdem wurden für SOAP RPC einige neue Fault Codes hinzugefügt.
Bei dem Aufbau von SOAP Faults gibt es auch einen großen Unterschied. In Version 1.1 war es üblich, dass ein Faultcode mittels sogenannter Punktnotation erweitert wird. Es wird an den Faultcode einfach ein Punkt und dann ein selbstdefinierter Fehlercode angehängt, um eine Fehlerhierarchie zu schaffen. Diese Repräsentation wurde durch eine XML-konformere ersetzt. Die Elemente `faultcode` und `faultstring` wurden in `env:Code` und `env:Reason` umbenannt. Nähere Informationen zum Thema SOAP Faults in SOAP Version 1.2 entnehme man Abschnitt 2.4.4.
In SOAP 1.2 wird ein `env:NotUnderstood` Header Element eingeführt, dass anzeigt, welcher SOAP Header Block vom Empfänger nicht verarbeitet werden konnte, wenn ein `env:MustUnderstand` Fault Code auftritt. In SOAP 1.1 gibt es zwar den Fault Code, aber eine genaue Beschreibung, an welchem SOAP Header die Bearbeitung gescheitert ist, gibt es nicht.
Dadurch, dass SOAP Version 1.2 als XML Infoset definiert wurde, hat das Attribut `env:mustUnderstand` eines SOAP Header Blocks entweder den Wert „true“ oder „false“. Früher hatte dieses Attribut den Wert „0“ oder „1“. Wir

werden auf diese Thematik im Abschnitt zur Interoperabilität von SOAP Implementierungen noch näher eingehen.
In der neuen Version von SOAP gibt es das Attribut `env:relay` für einen Header Block, um anzuzeigen, ob dieser Header Block weitergesendet werden soll, obwohl er vom empfangenden SOAP Knoten gar nicht verarbeitet worden ist.

2.12.2 Änderungen im HTTP Binding

Im HTTP Binding zu SOAP 1.2 wurde der HTTP Header `SOAPAction` entfernt. Dazu wurde von der IANA ein neuer HTTP Statuscode 427 angefordert, der anzeigen soll, dass der Server einen solchen HTTP Header erfordert. Der Inhalt dieses Headers wird im optionalen Parameter `action` des „application/soap+xml“ Medientyps hinterlegt.
Der „Content-type“ Header soll „application/soap+xml“ lauten und nicht mehr „text/xml“ wie in der Vorgängerversion. Genauere Informationen zu diesem neuen Medientyp findet man in [6].
SOAP Version 1.2 enthält eine genauere Beschreibung, wie HTTP Statuscodes bei den verschiedenen SOAP Faults anzuwenden sind und wie die Beziehung von Fehlercode zu SOAP Fault ist.
Es wurde ein zusätzliches Message Exchange Pattern definiert, das in Verbindung mit einem HTTP GET ein idempotentes und sicheres Abfragen von Informationen gewährleisten soll.

2.12.3 Änderungen am SOAP RPC

In SOAP 1.2 gibt es ein `rpc:result` Element, das eine Referenz auf den Rückgabewert des SOAP Remote Procedure Calls enthält. Tabelle 5 zeigt eine Gegenüberstellung der beiden Versionen im Bezug auf den Rückgabewert eines RPC.
Wie bereits weiter oben erwähnt wurden in der neuen Version von SOAP mehrere Fault Codes für SOAP RPC definiert. Weitere Informationen zu SOAP RPC in Version 1.2 findet man im Abschnitt 2.9 in diesem Dokument.

2.12.4 Änderungen am SOAP Encoding

Das SOAP Encoding der neuesten Version von SOAP soll eine Vereinfachung gegenüber der Vorversion bringen.
Für SOAP 1.2 wurde ein abstraktes Datenmodell, das auf einem beschrifteten, gerichteten Graphen basiert, entworfen. Das SOAP Encoding hängt von dieser Darstellung der Daten ab. Die Unterstützung der Datenkodierung der SOAP Version 1.2 ist optional.

SOAP 1.1	SOAP 1.2

`<env:Body>`
`  <m:addiereZahl>`
`  <m:Ergebnis>`
`  27`
`  </m:Ergebnis>`
`  </m:addiereZahl>`
`</env:Body>`
	`<env:Body>`
	`  <m:addiereZahl>`
	`  <rpc:result>`
	`  m:Ergebnis`
	`  </rpc:result>`
	`  <m:Ergebnis>`
	`  27`
	`  </m:Ergebnis>`
	`  </m:addiereZahl>`
	`</env:Body>`

Tabelle 5: Gegenüberstellung der Rückgabe eines SOAP RPC

SOAP 1.1	SOAP 1.2

`<Feld`
`  enc:arrayType="xs:int[2]>`
`  <Nummer>1</Nummer>`
`  <Nummer>2</Nummer>`
`</Feld>`
	`<Feld`
	`  enc:itemType="xs:int"`
	`  enc:arraySize="2">`
	`  <Nummer>1</Nummer>`
	`  <Nummer>2</Nummer>`
	`</Feld>`

Tabelle 6: Gegenüberstellung von Array Serialisierungen

Die Serialisierung von Arrays wurde in Version 1.2 geändert. Eine Gegenüberstellung der beiden Versionen in Bezug auf die Darstellung von Arrays zeigt Tabelle 6. Außerdem gibt es in der neuesten SOAP Variante keine Unterstützung für teilweise übertragene oder sogenannte „spärliche“ (sparse) Arrays.
Die Referenzierung von Werten innerhalb einer SOAP Nachricht hat sich in der neuen Version von SOAP ebenfalls geändert. Statt des `href` Attributs in SOAP/1.1 gibt es nun ein Attribut `env:ref` vom Typ `IDREF`.
Eine Typisierung von Knoten einer SOAP Nachricht ist in SOAP 1.2 nur mehr optional.
Seit Version 1.2 gibt es ein Attribut `nodeType`, das optional ist und die Struktur des Knotens angibt. Wie bereits in Abschnitt 2.8.1 erwähnt, kann dieses Attribut nur die Werte „simple“, „struct“ oder „array“ besitzen.

2.12.5 Versionsmanagement

In [39] werden die Versionsmanagementregeln für einen SOAP Knoten beschrieben, wenn ein Knoten die Versionierung von SOAP 1.1 bis SOAP 1.2 unterstützt. Diese Regeln müssen dann entsprechend implementiert werden. Die Regeln sind wie folgt:

1. Ein SOAP Knoten, der SOAP 1.1 unterstützt, wird, wenn er eine SOAP Nachricht im Format der Version 1.2 empfängt, einen „Version Mismatch“ SOAP Fault generieren, der auf einem SOAP/1.1 Nachrichtenkonstrukt basiert. Dies bedeutet, dass der SOAP Envelope der SOAP Version 1.1 Nachricht einen lokalen Namen `Envelope` besitzt und im Namensraum „http://schemas.xmlsoap.org/soap/envelope/“ liegt.

2. Ein SOAP 1.2 Knoten, der eine SOAP/1.1 Nachricht erhält, hat zwei Optionen zur Auswahl. Diese wären:

   - Er kann die Nachricht als SOAP/1.1 Nachricht verarbeiten, wenn er diese SOAP Version unterstützt, oder
   - muss einen „Version Mismatch“ SOAP Fault basierend auf einer SOAP/1.1 Nachricht, die der SOAP/1.1 Semantik folgt, generieren. Der SOAP Fault sollte einen `Upgrade` SOAP Header Block enthalten, der anzeigt, dass SOAP Version 1.2 vom fehlergenerierenden Knoten unterstützt wird. Das erlaubt dem SOAP 1.1 Knoten eine korrekte Interpretation des SOAP Faults.

Listing 8 zeigt eine Antwort SOAP Nachricht eines SOAP Version 1.2 Knotens, der eine SOAP/1.1 Nachricht empfangen hat, die er nicht verarbeiten konnte und mit einem `env:Upgrade` SOAP Header Block quittierte.

```
<? xml version="1.0" ?>
<env:Envelope xmlns:env="http://schemas.xmlsoap.org/soap/envelope/">
  <env:Header>
    <env:Upgrade>
      <env:SupportedEnvelope qname="ns1:Envelope"
        xmlns:ns1="http://www.w3.org/2003/05/soap-envelope"/>
    </env:Upgrade>
  </env:Header>
  <env:Body>
    <env:Fault>
      <faultcode>env:VersionMismatch</faultcode>
      <faultstring>Version Mismatch</faultstring>
    </env:Fault>
  </env:Body>
```

```
</env:Envelope>
```

Listing 8: SOAP Version 1.2 Knoten, der einen SOAP Fault generierte, weil er eine SOAP 1.1 Nachricht erhielt.

2.13 SOAP und WSDL

Die „Web Services Description Language“, kurz WSDL genannt, dient als maschinenlesbare Schnittstellenbeschreibung für SOAP Webservices. Es handelt sich um ein XML Dokument, welches plattform- und programmiersprachenunabhängig Services beschreibt.
Ein WSDL-Dokument stellt alle Informationen zur Verfügung, die ein Client benötigt, um ein entferntes Service aufzurufen. In den meisten Programmiersprachenumsetzungen dient es dazu, um sich einen sogenannten Proxy für ein Service zu erstellen. Es kann sich je nach Implementierung um einen statischen oder dynamischen Proxy handeln. Dieser stellt dann die Funktionen des entfernten Dienstes zur Verfügung. Ebenso gibt es bei einigen Implementierungen die Möglichkeit, die extrahierten WSDL-Daten in ein `Call`-Objekt zu laden, in dem dann die entsprechenden Einstellungen für das Service bereits voreingestellt werden. Der Entwickler muss hier nur noch die aufzurufende Methode mit den Parametern bekanntgeben. So soll im Allgemeinen ein Funktionsaufruf beim entfernten Service gekapselt werden.
Eine genaue Beschreibung des Aufbaus eines solchen WSDL-Dokuments findet sich in [26].

2.14 Service Stile und Kodierung

Eigentlich setzt sich SOAP aus zwei Konzepten zusammen. Einerseits wäre dies der Stil (style), der entweder Dokument-orientiert oder RPC-orientiert ist. Andererseits gibt es auch noch das Encoding, das entweder „literal“ oder „encoded“ ist. Theoretisch lassen sich diese vier Möglichkeiten beliebig kombinieren, tatsächlich werden aber meist die Varianten „Document/literal“ und „RPC/encoded“ verwendet. Diese beiden Varianten geben an, wie SOAP Nachrichten zwischen zwei Kommunikationspartnern ausgetauscht werden.

2.14.1 Der Unterschied

„RPC/encoded“ erledigt eigentlich die gleichen Aufgaben, die schon mittels Technologien wie CORBA ([16], DCOM ([20]) oder Java RMI([37]) gelöst werden konnten. Diese SOAP Variante ist dafür ausgelegt, einen entfernten Prozeduraufruf zu ermöglichen. Dazu sind beliebig viele Parameter nötig. Im

Endeffekt liefert dieser Aufruf dann einen Rückgabewert. Die Daten werden mittels einem in der SOAP Spezifikation näher definierten Format kodiert, das einfach zu erstellen ist. Für erfahrene Entwickler ist diese Art der Nutzung von SOAP leicht zu verstehen. Auch die Erstellung von SOAP Implementierungen, die diesem Schema folgen, ist leicht zu bewerkstelligen.
Auch bei „Document/literal“ wird ein Prozeduraufruf durchgeführt. Der Parameter und auch der Rückgabewert einer solchen Prozedur oder Methode sind XML-Dokumente. Daher gibt es von dieser Seite her keine Einschränkung, wie komplex so ein Parameter oder Rückgabewert sein kann. Die Dokumente werden mittels einem XML Schema beschrieben und können somit validiert werden. Dies erhöht aber die Ansprüche an die SOAP Implementierungen. Diese müssen nun mit der zusätzlichen Komplexität umgehen können, um diese SOAP Variante korrekt verarbeiten zu können.
„RPC/Encoded“ findet in den meisten Java SOAP Implementierungen seine Umsetzung. Hier wäre vor allem Apache Axis zu nennen, das die Form des Nachrichtenaustauschs als Standardeinstellung benutzt. Microsoft setzt schon seit längerem standardmäßig auf das dokumentenbasierte Nachrichtenaustauschmodell.

2.14.2 Fazit und Ausblick

Dokumentorientierte Services zeichnen sich durch eine losere Bindung zwischen Aufruf und Antwort aus. Diesem Umstand ist es auch zu verdanken, dass asynchrone Protokolle beziehungsweise Funktionsaufrufe möglich sind. Weiters erlaubt dieses Nachrichtenaustauschmodell auch die Bearbeitung durch mehrere Parteien beziehungsweise die Aufteilung der Arbeit zu einem späteren Zeitpunkt.
Mittlerweile sind fast alle Verantwortlichen der führenden SOAP Implementierungen der Meinung, dass „Document/literal“ die Zukunft gehört ([60, 30]). Mit SOAP Version 1.2 ist man vom W3C an sich schon soweit, dass SOAP Kommunikationsplattformen die SOAP 1.2 unterstützen, nicht mehr notwendigerweise das von ihnen spezifizierte SOAP Encoding implementieren müssen. In der Spezifikation zu dieser SOAP Version wurde es explizit als optional gekennzeichnet. Durch diesen Umstand lässt sich schon der Trend hin zu dokumentorientierten Services ablesen.

3 Aktuelle Implementierungen

In diesem Kapitel sollen einzelne SOAP Implementierungen näher untersucht werden.
Interessant an diesem Kapitel wird sein, welche SOAP Features ihre Umsetzung gefunden haben, und wie diese in den einzelnen Implementierungen realisiert wurden. Es soll darauf eingegangen werden, welche Implementierungen die aktuelle SOAP 1.2 Spezifikation bereits umgesetzt haben.
Als erste Implementierung wird Apache Axis für Java näher unter die Lupe genommen werden. Danach folgt ein kurzer Einblick in die C++ Implementierung dieses Frameworks. Da diese Implementierung sehr stark auf die Java Version aufbaut, wird sich dieser Überblick nur auf grundsätzliche Dinge beschränken.
Weiters soll auch die Umsetzung von SOAP Nachrichten im .NET Framework untersucht werden. Es wird interessant sein, wie .NET, das ja von Microsoft entworfen wurde, die in [39] erarbeiteten „Empfehlungen“ umsetzt. Grundsätzlich gibt es hier zwei recht konträre Konzepte, wie auf SOAP Kommunikation zurückgegriffen werden kann. Zum einen gilt es in dieser Arbeit das ASP.NET Framework und zum anderen die .NET Remoting Architektur zu untersuchen.
Ein weiterer Abschnitt dieser Arbeit ist der Java XML RPC (kurz JAX-RPC) Spezifikation von Sun, die im Zuge des Java Community Process entstanden ist, gewidmet. Parallel zu dieser Spezifikation wurde nämlich auch eine Referenzimplementierung entwickelt, auf die in diesem Abschnitt noch gesondert eingegangen wird.
Hier soll auch nicht verschwiegen werden, dass Apache Axis ebenfalls die Schnittstellen dieser JAX-RPC Spezifikation implementiert hat. Dies soll ein größtmögliches Maß an Interoperabilität zwischen den einzelnen Java SOAP Implementierungen sichern.

3.1 Die Apache Axis SOAP Implementierung für Java

Als erste Implementierung wird hier Apache Axis („Apache eXtensible Interaction System“) für Java näher beschrieben. Die untersuchte Version dieses Projekts ist 1.1. Zur Zeit der Erstellung dieses Dokuments gab es schon eine Beta-Version 1.2. Diese soll in einem Ausblick am Schluss dieses Kapitels näher betrachtet werden.
Bei Apache Axis handelt es sich um eine Open Source Implementierung von SOAP. Diese unterliegt der Lizenz der Apache Software Foundation. Des Weiteren ist es mittels Axis möglich, Server, Clients und auch Gateways zu entwickeln. Das Hosting von Web Services, damit ist der Betrieb und

das Anbieten von Web Services gemeint, kann auch mit Axis Framework durchgeführt werden.

3.1.1 Historisches

Als „Urvater“ der Apache Axis Implementierung kann man ohne Umschweife „SOAP for Java“ (SOAP4J) von IBM nennen. Dieses Projekt schuf die erste Implementierung des SOAP 1.1 Standards. Mit diesem Projekt wurde eine solide Grundlage für Folgeprojekte geschaffen.
Im Jahr 2000 wurde das Projekt „SOAP4J“ aufgelöst und an die Apache Software Foundation übergeben. Dieser Umstand sollte sicherstellen, dass es Entwicklern von E-Business Anwendungen möglich ist, auf ein offenes, herstellerungebundenes System zurückzugreifen. Bei Apache bekam das Projekt den Namen Apache SOAP.
Gegen Ende des Jahres 2000 entwickelte sich aus dem Projekt Apache SOAP das Apache Axis Projekt.
Derzeit gilt Axis als die offenste SOAP Implementierung in Java.
Eine der ersten Aufgaben bei der Erstellung des Apache Axis Projekts war, dass die Architektur komplett überdacht wurde. So verwendet diese SOAP Implementierung SAX („Simple API for XML“) Filterketten und greift somit auf einen Event-basierten XML Parser zurück. Bei Apache SOAP wurde noch das DOM („Document Object Model“) bevorzugt. Diese Art der Implementierung hatte aber deutliche Leistungseinbußen zu verzeichnen.
In puncto Interoperabilität, Funktionalität und Performance hat Axis mit der jetzigen Version seine Vorgänger bereits übertroffen ([21]). Ein wesentlicher Vorteil von Axis ist die beharrliche Orientierung nach der JAX-RPC Spezifikation von Sun, die in Abschnitt 3.4 genauer behandelt wird.

3.1.2 Leistungsmerkmale

Die folgenden Merkmale sind bei der Axis Implementierung positiv aufgefallen.
Die ersten Kennzahlen dieser Implementierung aus [3] wären:

- **Lizenzbestimmungen**: Der gesamte im Apache Axis Projekt enthaltene Sourcecode ist Open Source unter der Lizenz der Apache Software Foundation. Es ist möglich, in die Implementierung Einsicht zu nehmen. Dies war für Untersuchungen bezüglich der Umsetzungen von einigen Funktionalitäten auch dringend notwendig, da die Dokumentation zu diesem Projekt zu wünschen übrig lässt.

- **Performancemerkmale**: Das Apache Axis Projekt setzt beim Verarbeiten von SOAP Nachrichten mit den SAX Filterketten auf einen Event-basierten Parser. Dies bedeutet, dass die XML Dokumente mit signifikant größerer Geschwindigkeit bearbeitet werden als in der Vorversion Apache SOAP. Dort wurde nämlich noch ein DOM („Document Object Model“) Parser verwendet, welcher die ganze Struktur der SOAP Nachricht in einen DOM-Baum abbilden musste.

- **Konfiguration und Erweiterbarkeit**: Axis bietet Entwicklern die Möglichkeit, Erweiterungen völlig frei in Axis zu integrieren. Hier sei vor allem die Bearbeitung von Headern und das System Management hervorgehoben.

- **unterstützte Transportprotokolle**: Bei der Konzeption des Apache Axis Projekts ist auf vollkommene Unabhängigkeit vom Transportprotokoll wert gelegt worden. Es können HTTP, JMS oder SMTP verwendet werden, die SOAP Engine ist aber vollkommen transportunabhängig definiert worden. Derzeit gibt es für Axis HTTP, JMS und experimentell auch SMTP als Transportprotokolle für SOAP Nachrichten. Erweiterungen sind bei dieser Implementierung jedenfalls möglich.

- **Stabilität**: Das Axis Projekt definiert eine bestimmte Menge von öffentlichen Schnittstellen, welche sich relativ langsam, im Vergleich zum Rest des Axis Frameworks, verändern.

- **unterstützte SOAP Versionen**: Derzeit findet sich bei Axis sowohl Unterstützung von SOAP 1.1 als auch teilweise von 1.2. Es bleibt anzumerken, dass SOAP/1.1 immer noch die Standardversion bleibt. In der aktuellen stabilen Version von Axis wird aber nicht die aktuelle Spezifikation von SOAP 1.2 umgesetzt sondern eine frühere, was an unterschiedlichen Namensräumen bei den SOAP Nachrichten zu erkennen ist.

- **WSDL Unterstützung**: Apache Axis unterstützt die Version 1.1 der „Web Service Description Language“. Es ist möglich, sich mittels einer WSDL Beschreibung eines SOAP Services einen Proxy für das gegebene Service zu generieren. Ebenso wird vom Axis Framework die automatische Generierung einer WSDL Datei für ein mit Axis erstelltes Service bereitgestellt.

- **Unterstützung weiterer Spezifikationen**: In der vorliegenden Version 1.1 von Apache Axis wird die Spezifikation JAX-RPC Version 1.0

implementiert. Dies bietet ein einheitliches Interface für die Client- beziehungsweise Serverprogrammierung mit dem Axis Framework. Durch die Umsetzung der JAX-RPC Spezifikation ist es auch notwendig gewesen, den von Sun entwickelten Standard SAAJ („SOAP with Attachment API for Java“) umzusetzen. Durch Hinzufügen der Funktionalität dieses Standards ist es mittels Apache Axis möglich, auf SOAP Nachrichten und deren Attachments in MIME kodierten Nachrichten zuzugreifen.

- **Hosting der Services**: Normalerweise wird das Java Framework in einer Java Servlet Engine eingebettet, die das HTTP Handling für das Framework übernimmt. Typischerweise ist das Apache Axis Projekt perfekt auf den ebenfalls von Apache betreuten Apache Tomcat abgestimmt.

  Dies ist aber nicht die einzige Möglichkeit, Services der Öffentlichkeit zugänglich zu machen. Es gibt noch einen im Apache Axis Projekt befindlichen Standalone-Server, der die Implementierung eines simplen HTTP-Servers darstellt. Dieser Server hat die Aufgabe, an ihn gerichtete HTTP Requests zu übernehmen und diese dann an eine Axis Engine zu übergeben. Zu diesem sogenannten `SimpleAxisServer` sei aber angemerkt, dass dieser als nicht für Produktionszwecke einsetzbar gekennzeichnet ist.

- **SOAP Attachments**: Die Unterstützung der Spezifikation „SOAP with Attachment“, wie in [10] beschrieben, ist gegeben. Dies wurde im oben erwähnten Punkt durch die Umsetzung der SAAJ Spezifikation erreicht. Diese Verarbeitung von SOAP Attachments stellt die Standardeinstellung des Axis Frameworks dar. Mittels Axis hat man aber auch die Möglichkeit, DIME-kodierte Nachrichten zu verarbeiten. Dieser Standard wurde von Microsoft in [52] definiert.

- **unterstützte Nachrichtenmodelle**: Das Axis Framework bietet die Möglichkeit, es als Provider für SOAP RPC- oder dokumentbasierte SOAP Services zu verwenden. Ebenso wird literale oder SOAP Kodierung mittels Axis unterstützt. Bei der Unterstützung von dokumentbasierten Services steht man aber eher noch am Anfang. In einer kommenden Version soll diese aber ausgebaut werden.

3.1.3 Architektur

Axis basiert auf einigen Subsystemen, die im weiteren näher erläutert werden sollen.

Bei Axis geht es allgemein darum, Nachrichten zu verarbeiten. Es dreht sich alles um eine zentrale Verarbeitungslogik und um eine Folge von sogenannten „Handlern“, die in einer bestimmten Reihenfolge aufgerufen werden. Die genaue Ordnung wird durch zwei Faktoren bestimmt, nämlich der Deployment Konfiguration und ob die Axis SOAP Engine als Client oder Server aufgerufen wurde. In beiden Fällen wird zwischen den Handlern ein `MessageContext` Objekt ausgetauscht ([3]).
Ein solches `MessageContext` Objekt enthält folgende wichtige Teile:

1. eine „Request“ Message
2. eine „Response“ Message
3. eine Ansammlung von Eigenschaften

Es gibt zwei grundsätzliche Arten, in denen Axis ausgeführt werden kann:

1. Als Server, bei dem ein „Transport Listener“ einen `MessageContext` erzeugt und diesen dann dem Axis Verarbeitungsframework übergibt.
2. Im Client-Modus, bei dem Applikationscode einen `MessageContext` erzeugt und dann das Axis Verarbeitungsframework aufruft. Bei der Generierung des `MessageContext` Objekts wird empfohlen, das Axis Client Programmiermodell zu verwenden ([3]).

In jedem Fall ist es die Aufgabe des Apache Axis Frameworks, die erhaltenen `MessageContext`-Objekte durch eine konfigurierte Menge von Handlern zu schleusen. Jeder dieser Handler hat die Möglichkeit, den `MessageContext` auf eine vorbestimmte Art zu manipulieren.

3.1.3.1 Der Nachrichtenweg am Server

Der Weg einer Nachricht am Server wird in Abbildung 3 dargestellt.
Eine Nachricht gelangt mittels eines Transportprotokolls an einen Transport Listener. Es ist die Aufgabe dieses Listeners, die protokollspezifischen Daten in einem `Message` Objekt unterzubringen. Diese Nachricht wird dann in einem `MessageContext` abgelegt. Es werden vom Listener auch einige Eigenschaften des `MessageContext` gesetzt. Dann wird dieser `MessageContext` an die Axis Engine weitergegeben.
Die erste Aufgabe der Axis Engine ist es, die verwendete Transportmethode der SOAP Nachricht zu identifizieren. Ein Transport enthält eine Request Chain, eine Response Chain oder beides. Eine Chain ist ein Handler, der aus einer Folge von Handlern besteht, die aufeinander folgend aufgerufen werden.

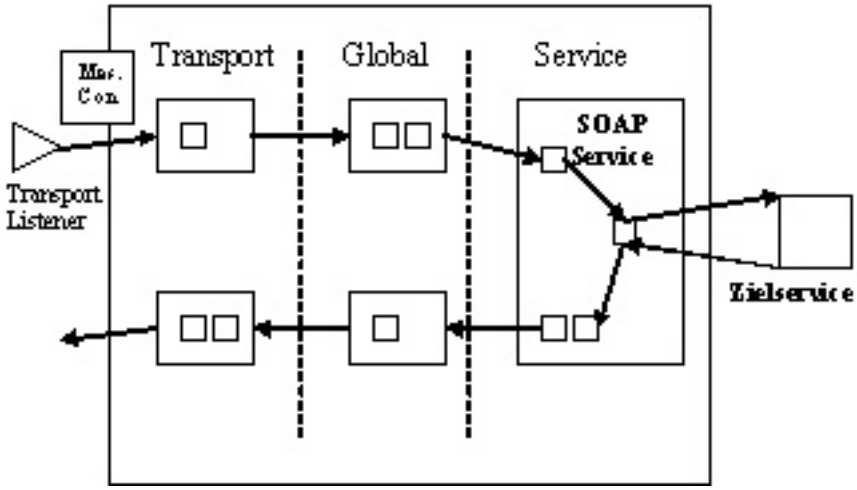

Abbildung 3: Weg einer Nachricht bei Benutzung des Axis Frameworks als Server

Wird ein solches `Transport` Objekt gefunden, so wird die `invoke()` Methode dieses Objekts mit dem `MessageContext` als Parameter aufgerufen. Im Zuge dessen werden alle Handler, die in der Request Chain dieses Transports definiert wurden, aufgerufen.
Nachdem alle Transport Request Handler abgearbeitet wurden, wird von der Axis Engine eine globale Request Chain lokalisiert. Wenn diese Chain konfiguriert wurde, dann werden alle darin enthaltenen Handler aufgerufen.
An einer Stelle der Abarbeitung sollte ein Handler das `serviceHandler` Feld des `MessageContext` Objekts gesetzt haben. Dieses Feld gibt an, welcher Handler aufgerufen werden muss, damit Service-spezifische Funktionalitäten zur Verfügung gestellt werden können. Ein Beispiel für solche Funktionalitäten wäre das Durchführen eines RPC Aufrufs an einem sogenannten „Back-end“ Objekt. Services in Axis sind typischerweise Instanzen der `SOAPService` Klasse (`org.apache.axis.handlers.soap.SOAPService`), die Request und Response Chains enthalten können. Services müssen einen sogenannten Provider enthalten, der einfach ein Handler ist, welcher die applikationsspezifische Logik enthält.
Für RPC-Style Aufrufe eines Axis SOAP Services gibt es mit der Klasse `org.apache.axis.providers.java.RPCProvider` bereits einen vorgefertigten Provider. Es handelt sich um einen Handler, der, wenn er aufgerufen wird, ein „Back-end“ Java-Objekt aufruft, dessen Klasse beim sogenannten „Deployment“ bestimmt wurde. Dieser Handler benutzt die SOAP RPC Konvention für die Bestimmung des Methodenaufrufs und überprüft, ob die XML-kodierten Argumente den Typen der benötigten Parameter der auszuführen-

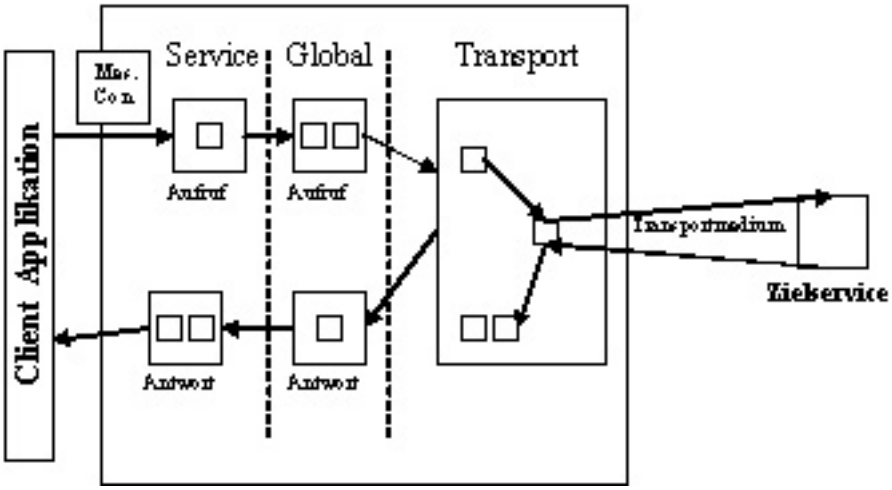

Abbildung 4: Weg einer Nachricht bei Benutzung des Axis Frameworks als Client

den Methode entsprechen. Das Back-end Objekt ist die konkrete Java-Klasse, die ein Service implementiert. Diese stellt schlussendlich die Funktionalität für das Service zur Verfügung.
Ein interessanter Aspekt des Axis Frameworks in Bezug auf das Deployment von Services ist das sogenannte Instant Deployment, das vom Framework zur Verfügung gestellt wird. So ist es möglich, dass man eine normale Java Source Klasse nimmt, sie mit der Dateiendung `.jws` versieht und sie in das Webapplikationen-Verzeichnis der Servlet Engine kopiert. Ab diesem Zeitpunkt übernimmt das Axis Framework diese Datei, kompiliert sie bei Bedarf und stellt sie als Service der Öffentlichkeit bereit. Das Axis Framework übernimmt alle Tätigkeiten, die mit dem Deployment des Services und dem Kompilieren der Source-Datei zu tun haben. Mit dieser Methode lassen sich aber in der Funktionalität nur sehr eingeschränkte Services realisieren. Zum Beispiel ist es nicht möglich, externe Klassen, die weder vom Servlet Container noch von der Java Laufzeitumgebung zur Verfügung gestellt werden, zu importieren.

3.1.3.2 Der Nachrichtenweg beim Client

Der Weg einer Nachricht am Client ähnelt stark dem am Server. Die Ausführungsreihenfolge wird aber umgedreht. Der erste Schritt ist, dass ein Service Handler, wenn dieser existiert, aufgerufen wird. Es gibt keinen Provider, weil das Service ja entfernt zur Verfügung gestellt wird. Es gibt aber immer noch die Möglichkeit von Request und Response Chains. Diese Service Request und Response Chains haben die Aufgabe, eine servicespezifische Verarbeitung der

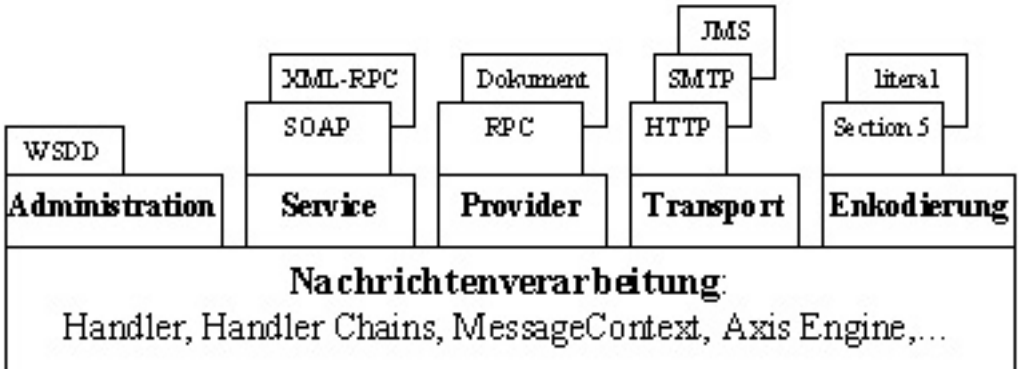

Abbildung 5: Die Subsysteme und ihr Layering bei Apache Axis

Request Message durchzuführen und auch die Response Message auf ihrem Weg zurück zu bearbeiten. Abbildung 4 zeigt den Weg einer SOAP Nachricht am Client.
Nach der Service Request Chain wird die globale Request Chain aufgerufen. Danach folgt der Transport der Nachricht. Dazu ist der Transport Sender vorgesehen. Dies ist ein spezieller Handler, dessen Aufgabe es ist, protokollspezifische Operationen durchzuführen, um eine Nachricht zum und vom Ziel-SOAP-Server zu bekommen. Die Antwort, wenn es überhaupt eine gibt, wird im `responseMessage` Feld des `MessageContext`-Objekts untergebracht. Danach wird dieses Objekt durch die Response Chains geführt.

3.1.3.3 Die Subsysteme

Axis besteht aus einer Sammlung von verschiedenen Subsystemen, die mit dem Ziel zusammenarbeiten, die Verantwortlichkeiten exakt zu trennen und Axis modular zu gestalten. Subsysteme, die ordentlich auf ein Schichtenmodell (engl. „Layer“) abgestimmt sind, erlauben es Teile des Systems zu benutzen, ohne das gesamte System benützen zu müssen.
Abbildung 5 zeigt das Layering der einzelnen Subsysteme im Axis Framework. Die niedrigeren Layer sind unabhängig von den höheren Schichten. Die übereinander liegenden Rechtecke zeigen komplett unabhängige Alternativen an. Zum Beispiel sind die Transportmethoden HTTP, JMS oder SMTP von einander unabhängig definierte Subsysteme, die aber gemeinsam benutzt werden können.
Es bleibt aber anzumerken, dass der Sourcecode von Axis nicht so sauber in Subsysteme getrennt wurde, wie es hier in den Subsystemen beschrieben ist. Einige Subsysteme sind über mehrere Packages verteilt und andere Packages

überschneiden sich mit mehr als einem Subsystem. Eine Umstrukturierung des Codes wird erst in späteren Versionen zu erwarten sein.

3.1.3.4 Das Message Flow Subsystem

Dieser Absatz soll erläutern, welche Objekte notwendig sind, um den Weg einer SOAP Nachricht nachzuvollziehen. Das Subsystem, das für die Verarbeitung von SOAP Nachrichten zuständig ist, soll jetzt näher beschrieben werden.

3.1.3.4.1 Handler und Chains

Handler werden in einer bestimmten Reihenfolge aufgerufen, um eine Nachricht zu bearbeiten. An einem bestimmten Punkt in der Reihenfolge kristallisiert sich ein Handler heraus, der einen Request abschickt und eine Response erhält oder einen Request abarbeitet und eine Response generiert. So einen Handler nennt man dann „Pivot Point“ der Reihenfolge. Wie oben bereits erwähnt, sind Handler entweder global, Transport- oder Service-spezifisch. Die Handler dieser drei Ausformungen lassen sich zu Ketten (engl. Chains) kombinieren. So besteht die zusammenfassende Reihenfolge von Handlern aus genau drei Ketten: Transport, Global und Service.
Ein Webservice liefert nicht notwendigerweise immer eine Response Message. Response Handler sind aber trotzdem nützlich, auch wenn keine Response Message generiert wurde, um Timer zu stoppen oder Ressourcen aufzuräumen.

3.1.3.4.2 Der Nachrichtenkontext

Zentrales Objekt bei der Verarbeitung einer SOAP Nachricht durch das Axis Framework ist der Nachrichtenkontext, dargestellt durch die Klasse `MessageContext`. Jeder Nachrichtenkontext ist, wie bereits weiter oben beschrieben, mit einer Request Message und/oder einer Response Message assoziiert. Jedes `Message`-Objekt wiederum besteht aus einem sogenannten `SOAPPart` und einem `Attachments`-Objekt, das das `AttachmentPart` Interface implementiert.

3.1.3.4.3 Die Axis SOAP Engine

Das Apache Axis Framework besitzt eine abstrakte `AxisEngine` Klasse mit zwei konkreten Subklassen. Die eine ist die `AxisClient` Klasse, welche für die clientseitigen Handler zuständig ist. Die andere Klasse ist der `AxisServer`, welcher wiederum die serverseitigen Handler kontrolliert.
Im Axis Framework wird ein `EngineConfiguration` Interface bereitgestellt, welches die Möglichkeit bietet, Handler Factories und globale Einstellun-

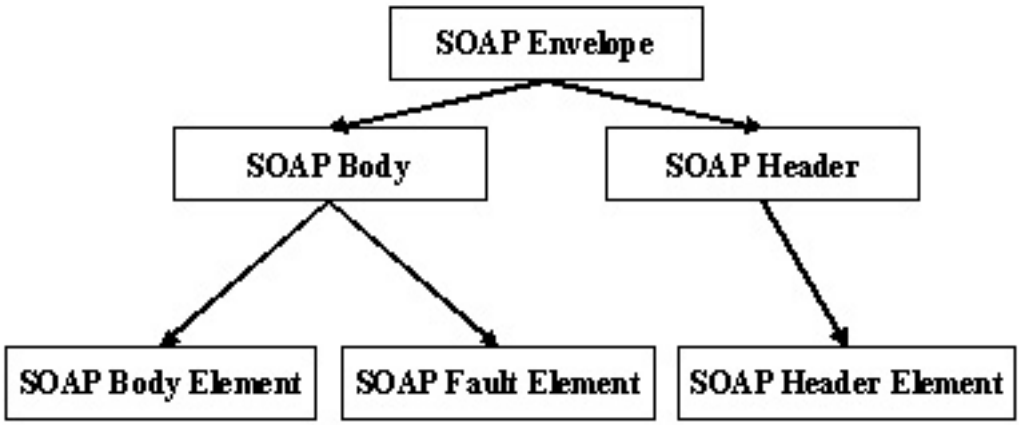

Abbildung 6: Aufbau der Baumstruktur einer SOAP Nachricht nach [3]

gen der Axis Engine zu konfigurieren. Die Engine erhält eine Referenz auf die `EngineConfiguration`, die während der Instanzierung der Axis Engine im Konstruktor übergeben wurde. Diese wird dazu benutzt, um Handler Factories zu erstellen oder globale Einstellungen auszulesen. Wird keine `EngineConfiguration` übergeben, so wird eine Standardkonfiguration angelegt. Die `AxisEngine` ist das Herzstück der Apache Axis Implementierung.

3.1.3.5 Der Aufbau von SOAP Nachrichten im Axis Framework

Die Klassen, die SOAP Nachrichten repräsentieren, formen eine Klassenhierarchie basierend auf der `MessageElement` Klasse, welche für Namensräume und Encodings zuständig ist. Das `SOAPHeaderElement` ist für die `actor` und `mustUnderstand` Attribute zuständig.
Während der Deserialisierung wird eine Baumstruktur aufgebaut, wie sie in Abbildung 6 schematisch dargestellt wird.

3.1.3.5.1 Deserialisierung einer SOAP Nachricht

Die Klasse, die hauptsächlich für das Verarbeiten von XML Dokumenten zuständig ist, ist `DeserializationContextImpl` (kurz „DCI“ genannt), welche die meisten ihrer Methoden vom Interface `DeserializationContext` ableitet. Die DCI managt die Konstruktion des Parsetrees und verwaltet einen Stack von SAX Handlern, eine Referenz zum `MessageElement`, das gerade bearbeitet wird, einen Stack von Namensraum Mappings, ein Mapping von IDs zu den zugehörigen Elementen, eine Menge von Typemappings für die Deserialisierung und einen SAX Event Recorder.
Zu Beginn der Abarbeitung einer SOAP Nachricht enthält der SAX Handler

Stack nur eine Instanz des `EnvelopeHandler`, welcher das Faktum darstellen soll, dass das Parsen des SOAP Envelope noch nicht begonnen hat. Der `EnvelopeHandler` wird mit einer Referenz auf einen `EnvelopeBuilder` konstruiert. Dieser `EnvelopeBuilder` ist für das Parsen des SOAP Envelopes zuständig.
Während des Parsens empfängt die DCI Events von dem SAX-Parser und verständigt entweder den SAX Handler, der an der obersten Stelle des Handler Stacks zu finden ist, den SAX Event Recorder oder beide.
Bei Start eines Elements ruft die DCI den SAX Handler an der obersten Stelle des Handler Stacks mit der Methode `onStartChild()` auf. Diese Methode gibt dann einen SAX Handler zurück, der dazu verwendet werden kann, das Kindelement zu parsen. Die DCI befördert diesen Handler ebenfalls auf den Stack. Danach wird die Methode `startElement()` aufgerufen. Unter anderem dient diese Methode dazu, ein neues `MessageElement` der entsprechenden Klasse zu erzeugen und alle Eltern-Kind Relationen der Baumstruktur herzustellen.
Am Ende eines Elements wird dann der SAX Handler von dem Handler Stack entfernt und die Methode `endElement()` aufgerufen. Es werden die SAX Handler für das Ende der Kindelemente aufgerufen. Zum Abschluss wird das `MessageElement`, welches gerade deserialisiert wird, auf das Elternelement des aktuellen Elements gesetzt.
Elemente, die nicht in der SOAP Spezifikation beschrieben wurden, werden mit einem `SOAPHandler` als SAX Event Handler und einem `MessageElement` als Knoten in der Baumstruktur verarbeitet.

3.1.3.6 Umsetzung des SOAP Encodings

Die Grundaufgabe des Encodings liegt darin, eine Transformation von Programmiersprachentypen in eine XML-konforme Darstellung zu bewerkstelligen. In Axis bedeutet dies, dass Java Objekte und Primitve in XML kodiert und XML in Java Objekte und Primitve dekodiert werden müssen. Die Basisklassen, die solche Dienste zur Verfügung stellen, sind Serializer und Deserializer.
Spezielle Serializer und Deserializer wurden geschrieben, um die spezifischen XML Verarbeitungsmechanismen wie DOM oder SAX zu unterstützen.
Jedes Tupel von Java Typ und XML Datentyp benötigt einen spezifischen Serializer und Deserializer. Darum ist es notwendig, dass ein Mapping zwischen den Tupeln zu den entsprechenden Serializer und Deserializer Factories angelegt wird. So ein Mapping wird auch als Type Mapping bezeichnet.
Es bleibt nur noch die Frage zu klären, welches Type Mapping für eine bestimmte SOAP Nachricht verwendet wird. Dieses Mapping wird durch das

Encoding, welches in der Nachricht angegeben wurde, bestimmt. Die sogenannte Type Mapping Registry enthält eine Auflistung der Encoding-Namen. Es wird die URI als Schlüssel in dieser Registry benutzt und liefert so das entsprechende Type Mapping für die Nachricht.
Zusammenfassend sei hier aufgelistet, was man wissen muss, um ein Java Objekt oder Primitiv in XML beziehungsweise einen XML Datentyp in ein Java Objekt oder Primitiv umzuwandeln ([3]):

- der Java Typ
- der XML-qualifizierte Name des Datentyps, der kodiert werden soll
- der XML Verarbeitungsmechanismus, der benutzt wird
- der Name des Encodings, welches für die Daten benutzt wird

Tabelle 7 zeigt eine Auflistung simpler XML-Datentypen mit ihrer Entsprechung in Java. Diese Aufzählung ist aus der JAX-RPC Spezifikation abgeleitet und entspricht auch der Umsetzung der Sun Referenzimplementierung, die in einem der folgenden Abschnitte näher untersucht wird.

XML Datentyp	**Java Datentyp**
xsd:base64Binary	byte[]
xsd:boolean	boolean
xsd:byte	byte
xsd:dateTime	java.util.Calendar
xsd:decimal	java.math.BigDecimal
xsd:double	double
xsd:float	float
xsd:hexBinary	byte[]
xsd:int	int
xsd:integer	java.math.BigInteger
xsd:long	long
xsd:QName	javax.xml.namespace.QName
xsd:short	short
xsd:string	java.lang.String

Tabelle 7: XML Datentypen und ihre Entsprechungen in Java bei Apache Axis

3.1.4 Zukünftige Entwicklungen

Zum Zeitpunkt dieser Arbeit steht bereits eine Beta-Version von Apache Axis 1.2 zur Verfügung. In dieser Testversion kommt ganz klar der Trend für die nächste stabile Version zum Ausdruck. Es sind eine bessere Unterstützung des „Document/literal“ Kommunikationsmodells mittels SOAP und eine weitreichendere Unterstützung des SOAP 1.2 Standards die Hauptentwicklungspunkte für die kommende Version.
Im Moment ist noch nicht klar, in welche Richtung sich die Unterstützung für SOAP 1.2 entwickeln soll. Es gibt Bestrebungen in der Axis Entwicklergemeinde das gesamte Versionsmanagement umzustrukturieren. Im Moment unterstützt die stabile Version von Axis eben nicht die finale Version der SOAP 1.2 Spezifikation. Dies hat sich aber bei der kommenden Betaversion schon geändert. Die Umsetzung des SOAP Versionsmanagements ist bei dieser Entwicklerversion noch identisch mit der von Axis 1.1.
Grundsätzlich wird eine Unterstützung beider Versionen angestrebt. Die Axis SOAP Engine soll dann je nach SOAP Version, die vom Aufrufer verwendet wurde, eine der Version des Aufrufers entsprechende SOAP Antwortnachricht generieren. Dieser Umstand soll sicherstellen, dass noch auf lange Sicht Abwärtskompatibilität zwischen den Axis Versionen und eben auch den SOAP Versionen gewährleistet wird.
Mit einer besseren Unterstützung des Dokumentenaustauschmodells will man der Kodierungsrichtlinie der .NET Implementierungen entgegenkommen. Es sind aber auch die Erwägungen aus Abschnitt 2.14 ein Beweggrund für eine verbesserte Unterstützung dieses Datenmodells. Zur Zeit ist die Unterstützung dieses Kommunikationsmodells nicht sehr stark ausgeprägt, sodass noch Handlungsbedarf in diesem Bereich der Implementierung besteht. Eine Unterstützung von „RPC/literal“ sowie „Document/literal“ ergibt sich ja gezwungenermaßen durch die Implementierung der JAX-RPC Spezifikation. In diesem Zusammenhang sei hier auch noch erwähnt, dass es dem Apache Axis Projekt auch noch ein großes Anliegen ist, die JAX-RPC Spezifikation 1.1 in der neuen Version umzusetzen. Wie bereits vorher erwähnt, baut die aktuelle stabile Version von Axis auf der Version 1.0 dieser Spezifikation auf. Apache Axis hat sich als eine verlässliche und stabile Basis für Services, die auf der Programmiersprache Java basieren, präsentiert. Es gibt aber immer noch Bereiche, an denen es etwas zu verbessern gilt beziehungsweise Fehler, die in einer neuen Version ausgebessert sein sollten.

3.2 Die Apache Axis SOAP Implementierung für C++

Dieses Kapitel hier sei der Umsetzung des Apache Axis Frameworks auf C++ gewidmet. Diese Implementierung stützt sich weitestgehend auf die Java Implementierung, soll aber durch die Umsetzung auf C++ Performancevorteile bringen, weil ja die Programmiersprache Java allgemein den Ruf hat, langsamer zu sein ([15]).

3.2.1 Historisches

Diese SOAP Implementierung wurde von einer Gruppe Softwareentwickler aus Sri Lanka geschaffen. Diese werden von einigen einheimischen Unternehmen unterstützt. Ziel war es, eine möglichst performante Version des Apache Axis Frameworks zu entwickeln ([4]).
Die erste Version dieser Implementierung erschien im Jahr 2003. Die Version zur Zeit der Erstellung dieses Dokuments lautet 1.1.1.

3.2.2 Die Umsetzung

Wenn man sich die Dokumentation und den Source-Code dieser SOAP Kommunikationsplattform genauer ansieht, merkt man, dass dieses Projekt sehr stark an das Java Projekt angelehnt wurde. Es wurde zwar versucht, möglichst alle grundlegenden Funktionen der Java Version nachzubauen, aber die Umsetzung erweckt den Eindruck, dass sie ziemlich stark für einen Zweck optimiert wurde. Das Handlerprinzip wurde zum Beispiel in der jetzigen Version noch nicht berücksichtigt. Es fehlt die damit verbundene Erweiterungsfähigkeit und Flexibilität bei dieser Implementierung.
Vom Hosting her gesehen, bietet diese SOAP Kommunikationsplattform einige Möglichkeiten. Grundsätzlich wird aber davon ausgegangen, dass diese Implementierung mit einem Apache Webserver benutzt wird. Es werden im Moment sowohl die Version 1.3 als auch die Version 2.0 dieses Webservertyps unterstützt. Die Axis Funktionalität wird über ein eigenes Apache Axis Modul dem Webserver zur Verfügung gestellt.
Auch in dieser Axis Implementierung gibt es einen `SimpleAxisServer`, der es ermöglicht, auf einfache Art und Weise Services der Öffentlichkeit zur Verfügung zu stellen. Dieser funktioniert ähnlich wie sein Java Pendant.
Die Nähe zum Java Projekt lässt sich durch den Umstand, dass es notwendig ist, die Java Bibliotheken in das Projekt einzubinden, verdeutlichen. Diese werden nämlich zur Generierung von WSDL-Beschreibungen der Dienste benötigt. Dieser Zweig wurde nämlich in der C++ Version der Axis Implementierung ausgespart und auf die Funktionalität der Java Implementierung zurückgegriffen.

Derzeit beschränkt sich der Umfang der vorhandenen Transportprotokolle auf HTTP. Es ist auch keine Unterstützung von anderen Transportprotokollen geplant. Dies zeigt deutlich den Charakter dieser SOAP Implementierung. Sie ist nämlich rein auf die Umsetzung von Web Services über HTTP konzipiert worden.

3.2.3 Zukünftige Erweiterungen

Es gibt aber innerhalb des Projekts Bestrebungen, diese SOAP Implementierung auch über Microsofts Internet Information Server (IIS) zu hosten ([4]). Die Umsetzung dieses Ziels schreitet bereits voran. Leider fehlt aber die notwendige Dokumentation zu diesem Teil des Projekts. Es ist ebenfalls geplant, dass man später die erstellten C++ Services auch über einen Jakarta Tomcat Servlet Container der Öffentlichkeit zur Verfügung stellt. Für diesen Teil des Projekts gilt das Gleiche, wie für die oben genannte Umsetzung im IIS, es fehlt an der nötigen Dokumentation für dieses Feature.
Auch im Bereich der WSDL Unterstützung sind Erweiterungen geplant. So soll es später möglich sein, auch ohne die zusätzlichen Axis Java-Bibliotheken WSDL-Beschreibungen zu generieren beziehungsweise zu verarbeiten.

3.2.4 Das Resümee

Meiner Meinung nach handelt es sich bei dieser Implementierung um eine auf ein Ziel optimierte Umsetzung des Java Projekts. Die neuralgischen Punkte bezüglich Performance wurden hier berücksichtigt und mittels C++ umgesetzt. Ein Punkt, der mir bei dieser Implementierung abging, war eine weitreichendere Unterstützung der Axis Architektur, wie es die Java-Implementierung vorzeigte.
Leider blieb auch die Dokumentation des Source-Codes hinter den Erwartungen zurück. Es gibt zwar eine sehr anschauliche Darstellung, wie man seine in C++ entworfenen Services der Öffentlichkeit zur Verfügung stellt, aber keine darüber hinausgehenden Informationen, wie sich zusätzliche Features umsetzen beziehungsweise einsetzen lassen.

3.3 .NET und SOAP

C# und viele andere von Microsoft konzipierte Programmiersprachen bauen auf die Funktionalitäten des Microsoft .NET Frameworks auf. Dies gilt auch für den Bereich der Web Services, die wiederum auf die Funktionalität des SOAP Stacks zurückgreifen, um Dienste zur Verfügung stellen zu können.

Eine weitere Methode, die in .NET Framework vorhanden ist, um SOAP Kommunikation zu nutzen, ist das sogenannte Remoting.
Zur Zeit der Erstellung dieser Arbeit war das .NET Framework in der Version 1.1 das aktuellste Paket. Nach einigen Recherchen stellte sich heraus, dass diese Version nur SOAP 1.1 unterstützt ([7]).
Da es sich bei diesem Framework um ein gern genommenes Instrument zur Erstellung von SOAP Diensten handelt, wollen wir es in diesem Kapitel genauer vorstellen.

3.3.1 Die SOAP Implementierung von .NET

Wie bereits oben erwähnt, verfügt das .NET Framework in der aktuellen Version über eine SOAP/1.1 Implementierung. Eine Implementierung der Version 1.2 wurde wieder aus dem Paket genommen, da es zur Zeit der Freigabe von .NET 1.1 noch keine offizielle Spezifikation von SOAP 1.2 gab. Die aktuelle SOAP Spezifikation wird erst in .NET 2.0 ihre Umsetzung finden ([49]).
Das .NET Framework wurde mit Blickpunkt auf die Erstellung von Webservices konzipiert. Die SOAP Implementierung galt als eines der Kernpunkte der neuen Technologien, die unterstützt werden sollten. Dies ist nur natürlich, da Microsoft als einer der Hauptvertreter gilt, die die SOAP Standardisierung vorantreiben.
Es gibt beim Microsoft .NET Framework zwei grundverschiedene Einrichtungen, die SOAP als Kommunikationsmittel verwenden. Zum einen wäre das der SOAP Stack des ASP.NET Web Service Zweiges, wo SOAP vor allem als ein Kommunikationsmittel zwischen einer Web Service Anwendung und einem SOAP Client über HTTP gesehen wird. Zum anderen gibt es da noch die .NET Remoting Architektur, die zum Austausch von Objekten zwischen zwei .NET Plattformen oder Prozessen dient. Diese beiden Technologien sind als konträr zu betrachten. In den folgenden Abschnitten sollen die etwaigen Unterschiede dieser beiden SOAP Kommunikationsprovider genauer untersucht werden. Beginnen wollen wir erst einmal mit dem ASP.NET Framework.

3.3.2 SOAP und ASP.NET Web Services

Mit der ASP.NET-Umgebung des .NET Frameworks hat man die Möglichkeit, SOAP/1.1-basierte XML Web Services zu erstellen, zu betreiben und in Client-Anwendungen zu benutzen.
Bei der Serialisierung von Objekten, die bei einem Web Service als Parameter beziehungsweise Rückgabewert gebraucht werden, dient die Klasse `XmlSerializer`. Die Klasse und deren genauere Architektur wird in die-

sem Abschnitt näher beleuchtet. In [63] findet man nähere Informationen zu ASP.NET und der `XmlSerializer` Klasse.
Diese Klasse dient dazu, um Objekte in eine entsprechende XML-Darstellung zu bringen. Als Anhaltspunkt bezüglich der Typumsetzung dient die XML Schema Spezifikation in [12]. Auf diese Weise soll ein größtmögliches Maß an Interoperabilität gewährleistet sein, da versucht wird, sehr nahe an dieser Spezifikation zu bleiben. Dies ergibt fast logischerweise, dass die Grundeinstellung für SOAP Services des ASP.NET die „Document/literal“ Kodierung von Objekten ist. Denn nur mit dieser Kodierung ist eine exakte Übereinstimmung mit dem XML Schema möglich.
In Tabelle 8 wird die Umsetzung der Datentypen der XML Schema Spezifikation in [12] mittels CLR Datentypen des .NET Frameworks aufgelistet. Ein vollständiges Mapping aller in der Spezifikation vorkommenden Datentypen auf die des .NET Frameworks findet sich in [48]. Einige der Einträge in dieser Tabelle werden uns in einem späteren Abschnitt nochmals begegnen.
In der Grundeinstellung von ASP.NET beschränkt sich die SOAP Kommunikation auf Austausch von XML Dokumenten mit HTTP als Transportprotokoll. Standardmäßig wird nämlich das RPC Encoding von SOAP nicht unterstützt. Laut Microsoft gibt es dafür triftige Gründe, denn das vorgeschriebene SOAP Encoding soll die Interoperabilität einschränken, wobei XML Dokumente, die keinem speziellen Encoding entsprechen, von jeder Applikation verarbeitet werden können sollten ([30], [60]). Dies heißt aber nicht, dass mit dem SOAP Stack von .NET keine SOAP RPC enkodierten Nachrichten verschickt werden können. Zu diesem Zweck gibt es ein Attribut, dass angibt, ob eine Methode RPC- oder Document-Style benutzt werden soll. Eine Unterstützung von „RPC/literal“ ist in ASP.NET nicht implementiert worden.

3.3.2.1 Der Weg einer SOAP Nachricht in ASP.NET

ASP.NET wird mittels des Konzepts des Pipelinings am Internet Information Server (kurz IIS) realisiert. In einer solchen Pipeline werden mehrere Module abgearbeitet.
Um SOAP Anfragen beantworten zu können, wird eigens ein SOAP Handler installiert, der diese Art von HTTP Nachrichten behandeln kann. Abbildung 7 zeigt wie ein HTTP Request in einem IIS abgearbeitet wird.
Mittlerweile ist aber ASP.NET nicht mehr nur auf dem von Microsoft implementierten Internet Information Server zu finden, sondern kann auch in Verbindung mit einem Apache Webserver benutzt werden. Vielversprechend in diesem Zusammenhang ist auch das Open Source Projekt Mono, das versucht, das .NET Framework auf Linux zu portieren. Mittels dieser Implementierung

XML Schema Typ	.NET Typ
anyURI	System.Url
base64Binary	System.Byte[]
Boolean	System.Boolean
Byte	System.SByte
Date	System.DateTime
dateTime	System.DateTime
decimal	System.Decimal
Double	System.Double
Float	System.Single
hexBinary	System.Byte[]
int	System.Int32
integer	System.Decimal
long	System.Int64
negativeInteger	System.Decimal
nonNegativeInteger	System.Decimal
nonPositveInteger	System.Decimal
positiveInteger	System.Decimal
QName	System.Xml.XmlQualifiedName
short	System.Int16
string	System.String
unsignedByte	System.Byte
unsignedInt	System.UInt32
unsignedLong	System.UInt64
unsignedShort	System.UInt16

Tabelle 8: Datentypen Unterstützung zwischen XML Schema Typen und .NET Framework Typen

soll es dann auch möglich sein, ASP.NET Services auf Linuxbasis anzubieten.

3.3.2.2 ASP.NET SOAP Extensions

Um direkten Zugriff auf eine „rohe“ SOAP Nachricht zu bekommen, müssen die sogenannten SOAP Extensions der eigentlich das Service implementierenden Klasse vorgeschalten werden. Die ASP.NET SOAP Extension Architektur wird in Abbildung 8 gezeigt. Diese Extensions bieten dem Entwickler eine erweiterte Zugriffsmöglichkeit sowohl auf eingehende als auch auf ausgehende SOAP Nachrichten.
Basis für die SOAP Extensions ist die SOAP Extension Klasse. Sie bietet die Möglichkeit, dass der Message Stream, der die SOAP Nachricht enthält, per Referenz übergeben wird. Eine beispielhafte Anwendung dieses Features

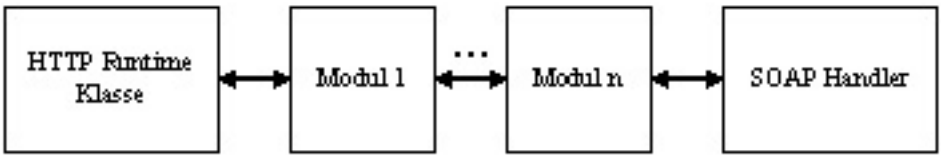

Abbildung 7: Pipeliningprinzip des ASP.NET Frameworks bei einem IIS

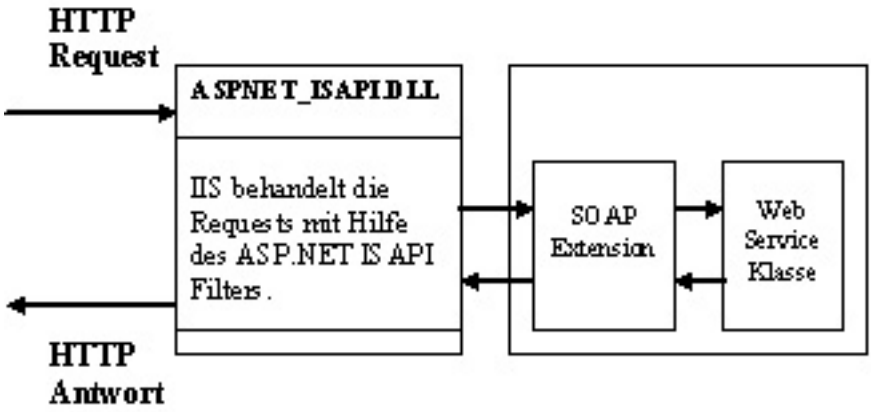

Abbildung 8: ASP.NET SOAP Extension Architektur

könnte unter anderem das Protokollieren der eingehenden und ausgehenden SOAP Nachrichten sein. SOAP Extensions haben die Funktion, dass sie direkt in den Datenstrom schreiben.
Eine SOAP Nachricht bei ASP.NET durchläuft mehrere sogenannte „SOAP Message Stages“. Diese beinhalten auf der Client- und auf der Serverseite mehrere Stadien. Diese wären jeweils Before Serialize, After Serialize, Before Deserialize und After Deserialize. Zu diesen Ereignissen lassen sich mittels des SOAP Extension Frameworks Erweiterungen für die jeweiligen Service-Klassen definieren.

3.3.2.3 Die ASP.NET XML Serialisierung

Wie bereits weiter oben erwähnt, übernimmt die Klasse `XmlSerializer` die Umwandlung von CLR (Common Language Runtime) Datentypen in XSD (XML Schema Datatypes) und umgekehrt. Es können nicht nur die in Tabelle 8 dargestellten simplen XML-Datentypen in einfache CLR-Datentypen konvertiert werden, sondern auch, wie in Tabelle 9 aufgelistet, einfache Typen, zusammengesetzte Typen, ja sogar eigene Klassen. Bei eigenen Klassen werden nur die öffentlichen Felder serialisiert. Daraus folgt, dass der `XmlSerializer`

Datentyp	Beschreibung
`XmlNode`	XML-Element
Felder von `XmlNode`	`XmlNode[]`
Einfache Typen	Beispiele: String, Int32, Byte, Boolean, Int16, Int64, Single, Double, Decimal, usw. (siehe Tabelle 8)
Felder von einfachen Typen und Aufzählungen	Felder (Arrays), Bsp.: String[] oder Integer[]
Klassen und Strukturen	Klassen und Strukturen mit öffentlichen Feldern oder Eigenschaften
Felder von Klassen/Strukturen	MyType[]
Aufzählungstypen	Public enum TestType {A=1, B=2}

Tabelle 9: Unterstützte Datentypen in ASP.NET über SOAP

keine exakte Kopie des Zustands eines Objekts erstellen kann, da er keinen Zugriff auf private Felder oder Eigenschaften einer Klasse hat.
Bei der Serialisierung mittels der Klasse `XmlSerializer` wird ein stark XSD-Datentypen zentriertes Modell verfolgt. Dies geschieht aus dem Grund, weil ein größtmögliches Maß an Kompatibilität zu anderen SOAP Implementierungen erreicht werden soll. Die Ausprägung des endgültigen Serialisierungsergebnisses lässt sich mittels Attributen in der Service Klasse steuern.
Mit dem `XmlSerializer` ist es auch möglich, Exceptions zu serialisieren. Diese Fehler werden in eine `System.Web.Services.Protocols.SoapException` eingebettet, welche dann mittels des `XmlSerializer` in einen SOAP Fault serialisiert wird.

3.3.2.4 Die Architektur des `XmlSerializer`

Es ist mit der `XmlSerializer` Klasse möglich, öffentliche Eigenschaften und Felder zu serialisieren. Die Typintegrität und weitere Informationen wie zum Beispiel Assembly-Daten werden nicht beibehalten.
Beim `XmlSerializer` handelt es sich um eine zentrale Klasse, über die sämtliche Serialisierungen abgehandelt werden. Durch verfügbare Attribute im gleichen Namensraum lässt sich der `XmlSerializer` noch zur Laufzeit nach eigenen Anforderungen umkonfigurieren.
Die beiden am häufigsten gebrauchten und auch wichtigsten Methoden in der Klasse `XmlSerializer` sind `Serialize()` und `Deserialize()`. Diese bei-

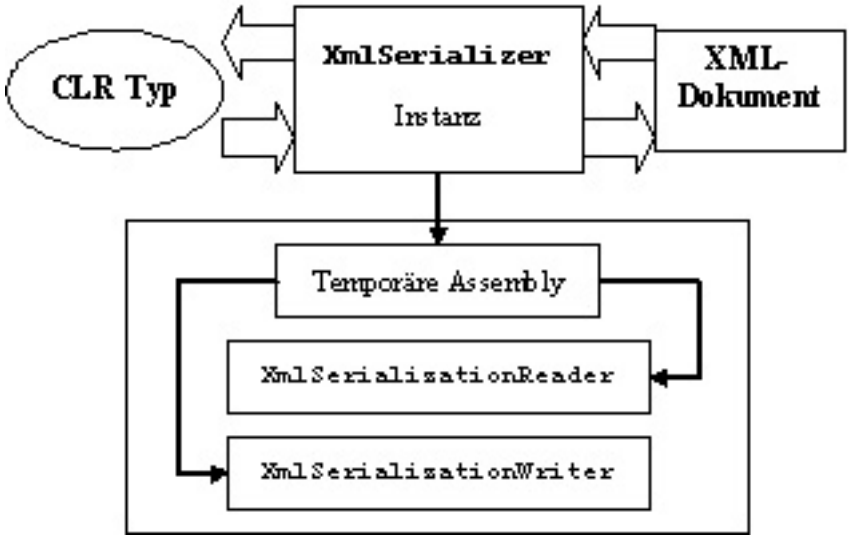

Abbildung 9: Dynamischer Aufbau eines `XmlSerializer`s.

den Methoden sind dafür zuständig, dass ein XML-Datentyp in einen CLR-Datentyp umgewandelt wird, beziehungsweise umgekehrt, dass ein CLR-Datentyp in einen XML-Datentyp überführt wird. Der `XmlSerializer` serialisiert aber nur Klassen, die einen Standardkonstruktor, dies bedeutet einen Konstruktor ohne Parameter, besitzen. Dieser Serializer kann keine Methoden serialisieren.
Mit dem `XmlSerializer` wird im ASP.NET Framework ein Serialisierungsmodell umgesetzt, dass durch die Serialisierung der öffentlichen Felder einer Klasse praktisch eine teilweise Kopie des Klassenobjekts in XML-Darstellung erzeugt. Die Serialisierung findet, wie bereits bei der SOAP Extension des ASP.NET Frameworks erwähnt, über ein Stream-Objekt statt, das entweder im Speicher vorhanden ist oder auf eine Datei im Dateisystem zeigt.
Dem Konstruktor für einen `XmlSerializer` wird ein Parameter übergeben, der angibt, für welchen Typ diese Serializer-Instanz zuständig ist, denn pro CLR-Typ wird nur eine `XmlSerializer`-Instanz benötigt. Bei der Instanzierung eines `XmlSerializer` wird dynamisch eine temporäre Assembly erzeugt. Dies geschieht aber nur, wenn sich noch keine Assembly für den übergebenen Typ im Cache befindet.
In der Assembly eines `XmlSerializer`s werden zwei Klassen referenziert. Diese sind einerseits vom Typ `XmlSerializationReader` beziehungsweise andererseits vom Typ `XmlSerializationWriter` und dienen zur Umwandlung von XML-Daten in CLR-Typen beziehungsweise CLR-Typen in XML-Daten. Der Code für diese beiden Klassen wird im Laufzeitcache abgelegt, auf

den dann die `XmlSerializer` Objekte zugreifen können. Eine schematische Übersicht zeigt Abbildung 9.
Der `XmlSerializer` bietet aber auch dem Entwickler die Möglichkeit, den Serialisierungsvorgang eines Services zu beeinflussen. Wie bereits weiter oben erwähnt, gibt es Attribute, die die Serialisierung von eigenen Klassen entscheidend beeinflussen.

3.3.3 .NET Remoting

Im Vergleich der beiden SOAP-Kommunikationsmodelle stellt Remoting das Konzept zur Verfügung, um ganze .NET Objekte samt ihrem aktuellen Zustand zu serialisieren. Deshalb wurde bei dieser Implementierung auf Interoperabilität wenig Wert gelegt, dadurch können aber auch entsprechend mehr Datentypen serialisiert werden, als es mit der `XmlSerializer` Klasse der Fall ist.
Während die SOAP-Anwendung von ASP.NET auf eine möglichst interoperable SOAP-Kommunikation über HTTP ausgelegt ist, erhält man mittels Remoting die Möglichkeit, über verschiedenste Transportprotokolle SOAP-Nachrichten zu verschicken.
Bei der Entwicklung von Remoting stellten sich die folgenden Anforderungen an das Team bei Microsoft ([63]):

- Unterstützung mehrerer Protokolle
- Unterstützung mehrerer Datenformate
- Verwaltung der Lebenszeit eines Objekts durch den Client oder Server
- Übergabe der Objektdaten per Referenz oder in einer Kopie
- Möglichkeit, Ereignisse und Rückrufe zu implementieren
- Allgemeine Erweiterbarkeit gemäß dem Kontextprinzip
- Interoperabilität mit anderen Plattformen, wobei dieser Punkt etwas zu kurz gekommen ist

In erster Linie dient .NET Remoting zum Austausch von Daten zwischen verschiedenen sogenannten Applications Domains. Diese stellen eine Abstraktion von physikalischen Prozessen dar. Die erwähnten Daten werden als Nachrichten zwischen den beteiligten Remoting-Parteien versandt. Beim Format der Nachrichten gibt es in der gegenwärtigen Implementierung zwei verschiedene sogenannte Formatter, nämlich den `SOAPFormatter`, der Nachrichten

als SOAP-Nachrichten formatiert, und den `BinaryFormatter`, der die Nachricht als Binärdaten in einem proprietären Format formuliert. Standardmäßig wird der `SOAPFormatter` über einen HTTP Channel eingesetzt und der `BinaryFormatter` über einen TCP Channel. Wie bereits oben erwähnt, lässt sich diese Grundeinstellung aber beliebig verändern. Es sei darauf hingewiesen, dass in der Basisimplementierung des .NET Frameworks nur HTTP und TCP als Transportkanäle zur Verfügung stehen. Weitere mögliche Transportmittel, wie etwa Message Queues, lassen sich aber leicht an das Remoting Konzept anpassen und mittels SOAP-Nachrichtenaustausch erweitern.
Zum Zugriff auf Daten über Applications Domains hinweg wird das Konzept des Marshaling benutzt. In .NET Remoting werden zwei Typen des Marshaling unterstützt. Diese wären zum einen die Standardeinstellung bei Remoting Marshal by Value (MBV) und zusätzlich Marshal by Reference (MBR). Klassen lassen sich per Attribut in den entsprechenden Typ überführen.

3.3.3.1 Die Architektur von .NET Remoting

Wie bereits in Abbildung 10 dargestellt, kommunizieren der Client beziehungsweise der Server mit einem Proxy beziehungsweise einem Dispatcher. Der Proxy stellt für den Client die Methoden der Server-Klasse zur Verfügung, enthält aber keine konkrete Implementierung dieser Methoden, sondern dient lediglich dazu, um auf die entfernten Methoden des Servers zuzugreifen. Der Dispatcher wiederum hat beim Server eine ähnliche Funktionalität. Er muss dafür Sorge tragen, dass die eingehenden Parameter-Daten in entsprechende Form gebracht werden, damit sie die Server-Klasse dann auch weiterverarbeiten kann.
Hauptelement aus SOAP-Sicht ist der Formatter, der bei SOAP als Nachrichtenprotokoll `SOAPFormatter` genannt wird. Der Formatter an sich dient dazu, die Objekte und Daten in ein serialisierbares Format zu transformieren beziehungsweise serialisierte Daten wieder in Objekte umzuwandeln. Als Ausgabe liefert der Formatter einen Stream. Der besagte Stream wird dann über den Channel geschickt. Die Formatter-Objekte werden dynamisch von der Channelimplementierung erzeugt. Umgesetzt wird dies dadurch, dass ein Channel zur Laufzeit einen Formatter anfordert, damit eine Nachricht umgewandelt wird. Das Zusammenspiel zwischen den Channels und den Formattern ist sehr feinkörnig konfigurierbar. Ebenso ist es dann möglich, das serialisierte Objekt aus dem Stream mittels des `SOAPFormatter`s zu extrahieren. Dazu wird die `Serialize`-Methode verwendet, die als Parameter einen Stream verlangt.
Mit dem `SOAPFormatter` ist es möglich, wirklich das gesamte Objekt zu serialisieren. Es werden auch die privaten Felder einer Klasse berücksichtigt. Die

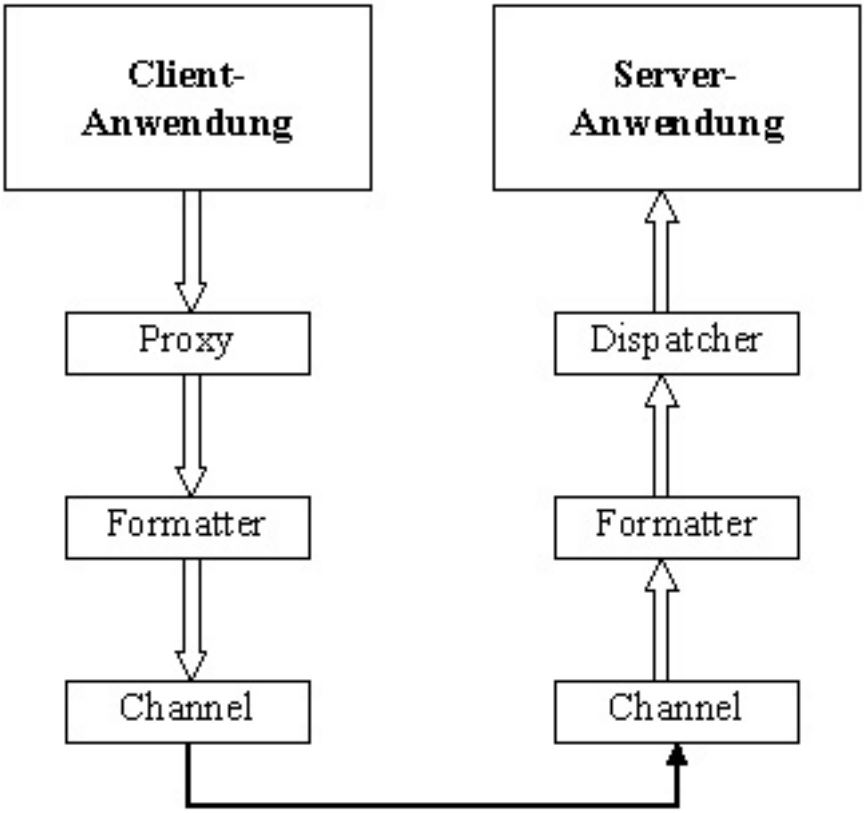

Abbildung 10: Überblick über die .NET Remoting-Architektur

Serialisierung eines Objekts mittels dem `SOAPFormatter` ist sehr interessant, da sich ja SOAP nicht wirklich für Marshal by reference eignet. Diese Form der Übergabe eines Objekts ist eigentlich in der SOAP Spezifikation nicht vorgesehen, wird aber von der .NET Remoting Architektur trotzdem dazu genutzt. Interessanterweise wird ein Teil der Informationen, die zur Deserialisierung eines Objekts benötigt werden, im Namensraum des serialisierten Objekts in der SOAP Nachricht gespeichert. So besteht ein Teil des Namensraums etwa aus der Tatsache, dass es sich bei dem serialisierten Objekt um eine .NET Klasse handelt. Es wird auch die gesamte Information über die Assembly in diesem Namensraum kodiert. Anhand dieser Daten lässt sich das Objekt, das bearbeitet worden ist, bestimmen. Dass diese Daten außer für eine andere Remoting Instanz schwer zu deuten sind, ist unmittelbar einsichtig. Diese Art von Parameterübergabe ist im Sinne der Interoperabilität nicht benutzbar.
Einen Punkt bezüglich des `SOAPFormatters` gilt es noch zu berücksichti-

gen. Diese Instanz hat Zugriff auf alle Daten eines Objekts, da es ja auch private Felder einer Klasse serialisieren kann. Aus diesem Grund muss der `SOAPFormatter` auch mit mehr Freiheiten was die Sicherheit und Vertraulichkeit von Daten betrifft ausgestattet werden. Implementierungstechnische Details einer Klasse wären leicht auszumachen, wenn man die SOAP Kommunikation zwischen Client- und Serveranwendung mitprotokolliert.
Es können auch eigene Formatter programmiert werden. Diese Formatter müssen das Interface `IRemotingFormatter` implementieren. So kann zum Beispiel ein Formatter für die SOAP Version 1.2 selber implementiert werden. Natürlich ist es auch möglich, dass ein Stream mittels Formatter wieder in das ursprüngliche Nachrichtenformat konvertiert wird.
Übertragen werden die durch den Formatter erzeugten Nachrichten über einen Channel. Dieser ist immer an eine bestimmte Protokollimplementierung gebunden. Im Moment werden vom .NET Framework, wie bereits früher schon erwähnt, nur TCP und HTTP Implementierungen angeboten. Eine Erweiterung der Transportmöglichkeiten für SOAP Nachrichten ist aber durchaus möglich. Diese selbst implementierten Channels lassen sich entsprechend leicht in die .NET Remoting Architektur integrieren.
Die gesamte Architektur des .NET Remoting ist mehrschichtig angelegt. Es ist aber Sorge getragen worden, dass ein Entwickler je nach Anforderung seine eigenen Implementierungsideen einbringen kann. Dies geschieht dadurch, dass der Entwickler bestimmte Teile der Architektur durch eigene Implementierungen ersetzen kann. So gibt es etwa die Möglichkeit einen eigenen Formatter zu entwerfen oder einen Transportkanal selber zu erstellen.
Da mit .NET Remoting über HTTP Channel und dem `SOAPFormatter` auch XML Web Services realisierbar sind, können SOAP-fähige Clients auch plattformübergreifend auf diese entfernten Services zugreifen. Es ist möglich, dass .NET Remoting eine WSDL Beschreibung des Services automatisch generiert. Doch es ist besonders darauf zu achten, dass bei .NET Remoting die Gewichtung hauptsächlich auf Umsetzung des CLR-Typsystems gelegt wurde. Wird zum Beispiel ein Datentyp `ArrayList` von einem Remoting Service als Rückgabewert definiert, so ist nicht sichergestellt, dass ein Client, welcher auf einer anderen SOAP Kommunikationsplattform aufbaut, diesen Datentyp auch verarbeiten kann, selbst wenn dieser in einer WSDL-Beschreibung angeführt wurde. Es kann nämlich durchaus sein, dass der Datentyp in einem Namensraum liegt, der der Client-Implementierung nicht bekannt ist.
Allgemein lässt sich aber sagen, dass der `SOAPFormatter` die gesamte SOAP 1.1-Spezifikation unterstützt und noch viel mehr CLR-Datentypen umwandeln kann als sein Gegenstück beim ASP.NET Framework. In diesem Zusammenhang bleibt anzumerken, dass es bei der Serialisierung von Objekten und Daten im Bereich das `SOAPFormatter`s nicht die Möglichkeiten der Ein-

flussnahme auf das endgültige Produkt der Serialisierung gibt, wie es mit den Attributen beim `XmlSerializer` sehr einfach zu bewerkstelligen ist.

3.3.4 XmlSerializer versus SOAPFormatter

Nach der Vorstellung der beiden SOAP-Kommunikationsmethoden des .NET Frameworks möchte ich in diesem Abschnitt die beiden Hauptkomponenten aus SOAP-Sicht kurz miteinander vergleichen.
In diesem Vergleich sollen Überlegungen aus den Bereichen Performance und auch Interoperabilität einfließen.
Grundsätzlich lässt sich zu diesen beiden Ansätzen sagen, dass das .NET Framework mit diesen beiden zwei total verschiedene Lösungen liefert, die einem Entwickler die Arbeit des Serialisierens eines Objekts abnehmen.

3.3.4.1 Interoperabilität

Die Designziele des `SOAPFormatter`s unterscheiden sich stark von denen des `XmlSerializer`s. Die Hauptaufgabe des letzteren ist die Umsetzung von .NET Klassen in XML-Datentypen, die durch ein XML Schema beschrieben werden. Der `XmlSerializer` gibt dem Entwickler aber auch die Flexibilität, in den Mapping Prozess des Umwandelns zwischen den beiden Datenrepräsentationsformen einzugreifen. Dies sichert Applikationen, die auf dem `XmlSerializer` aufbauen, aber auch ein gewisses Maß an Interoperabilität, sodass es möglich ist, mit anderen Systemen und Plattformen zu kommunizieren. Es wird auch klar, dass es sich bei diesem Serializer nicht um eine Allzweck-Serialisierungslösung handelt. Weiters bleibt anzumerken, dass ein Objekt oder Daten, welche mittels des `XmlSerializer` serialisiert beziehungsweise deserialisiert wurden, nicht mehr den selben Typ haben müssen, weil ja auf eine möglichst kompatible Umformung in einen XML-Datentyp gesetzt wurde.
Bei der Laufzeitserialisierung mittels `SOAPFormatter` wird aber auf eine ganz andere Richtung bei der Serialisierung von Objekten gesetzt. Dort ist es nämlich von immanenter Wichtigkeit, dass eine möglichst akkurate Beschreibung des serialisierten Objekts erzeugt wird. Die Einflussnahme auf den Serialisierungs- oder Deserialisierungsvorgang ist von geringer Bedeutung. Microsoft ist nämlich bei der Entwicklung des `SOAPFormatter` davon ausgegangen, dass dieser Formatter sowohl bei der Serialisierung als auch bei der Deserialisierung verwendet wird und somit auch keine Notwendigkeit für Interoperabilität zwischen verschiedenen Implementierungen besteht. Es wird davon ausgegangen, dass Applikationen, die mittels Remoting Daten austauschen, Zugang zu gleichen Assemblies haben.

3.3.4.2 Performance

Ein anderer Aspekt, wo diese beiden Lösungen sich merklich unterscheiden, ist die Performance bei den Serialisierungs- beziehungsweise Deserialisierungsvorgängen. Microsoft legte beim Entwurf des `XmlSerializer` besonderen Wert auf hohe Performance und verschob intensive Rechenoperationen in den Konstruktor. Der Konstruktor nämlich analysiert alle Typen, die er verarbeiten kann und kreiert intern kompilierte Klassen, die diese Typen dann sehr schnell verarbeiten können. Aus diesem Grund sind die beiden Methoden `Serialize` beziehungsweise `Deserialize` sehr schnelle Operationen, weil diese internen Klassen gespeichert werden und nicht für jedes Objekt neu erzeugt werden müssen. Es ist nicht notwendig den Typ des Objekts nochmals zu analysieren.
Der `SOAPFormatter` verhält sich da genau gegenteilig. Der Konstruktor benötigt keine Information über die Typen, die er verarbeiten soll. Dadurch muss der Formatter aber jedes Objekt nach der gegebenen Struktur analysieren. Bei Vergleichen dieser beiden Serialisierungsansätze stellte sich heraus, dass die Serialisierung von identischen Objekten mit einfacher Struktur mit dem `XmlSerializer` sechs mal schneller war als mit dem `SOAPFormatter` ([22]). Diese Performance-Kluft wurde mit der Komplexität der serialisierten Klasse immer größer.
Dies macht den `XmlSerializer` zu einem notwendigen Werkzeug für Applikationen, die auf einen hohen Durchsatz von Daten angewiesen sind, wie es nun einmal bei den ASP.NET Web Services der Fall ist.

3.3.5 Leistungsmerkmale

In diesem Kapitel sollen kurz die von der .NET Implementierung zur Verfügung gestellten SOAP Features und Erleichterungen für den Programmierer näher beleuchtet werden.
Eine Aufzählung der bereitgestellten Features wäre:

- **SOAP Version**: In der aktuell vorliegenden Version des .NET Frameworks wird nur die Version SOAP/1.1 unterstützt. Im Beta-Stadium der .NET 1.1 wurden auch kurzzeitig aktuelle Ausgaben der SOAP 1.2 Vorschläge umgesetzt, die aber in .NET 1.1 keinen Einzug gehalten haben.

  Vermutlich wird aber bereits die Version 2.0 des .NET Frameworks SOAP 1.2 unterstützen ([49]).

- **SOAP Attachments**: In der Basisausstattung des aktuellen .NET Pakets ist keine Unterstützung für SOAP Attachments vorgesehen. Es

wird aber als eine Art „Workaround“ ermöglicht, Binärdaten mittels Base64-Kodierung in einer SOAP-Nachricht zu übertragen. Dies stellt aber keinen adäquaten Ersatz für Attachments dar.

Ein Zusatzframework für auf .NET aufsetzende Web Services ermöglicht es aber, Konstrukte zu formulieren, die sowohl SOAP Nachrichten als auch Attachments enthalten. Bei diesem Zusatzpaket handelt es sich um die sogenannten Web Service Enhancements (kurz WSE genannt), die aktuell in der Version 2.0 vorliegen. Mit dieser Erweiterung für .NET Web Services ist es möglich SOAP Nachrichten mit Attachments zu verschicken. Es wird DIME als Format für diese Art von Nachrichten verwendet. Eine konkrete Beschreibung dieses Formats findet sich in [52]. Eine Einschränkung ist, dass sowohl der Client als auch der Server diese Erweiterung installiert haben müssen ([7]).

- **Zugang zu Implementierungsdetails**: Bei dieser SOAP Kommunikationsplattform handelt es sich um eine proprietäre Lösung der Firma Microsoft. Es nur erschwert möglich, eventuelle Implementationsfinessen näher zu untersuchen. Man ist in diesem Punkt sehr stark von der Dokumentation des Frameworks abhängig.

- **Kodierungseinstellungen**: Document Style ist eine Standardeinstellung bei der Erstellung eines Services mit dem ASP.NET Framework. Dies unterscheidet sich von den Standardeinstellungen vieler anderer SOAP-Implementierungen. In Verbindung mit den dokumentbasierten Nachrichtenmodell wird bei .NET auch immer die literale Kodierung benutzt. Das in der SOAP Spezifikation definierte Kodierungsmodell ist mit diesem Nachrichtenstil nicht kombinierbar.

  Mittels Attributen lässt sich dieses Verhalten aber ändern. Dies bedeutet, dass das .NET Framework auch SOAP Nachrichten im RPC Style erstellen kann. Es kann aber nur SOAP Kodierung benutzt werden. Eine Verbindung aus RPC und literaler Kodierung ist im .NET Framework nicht implementiert und kann auch deshalb nicht genutzt werden.

  Ganz anders verhält sich der Sachverhalt beim `SOAPFormatter` beziehungsweise .NET Remoting. Hier werden standardmäßig SOAP Nachrichten mit SOAP Encoding erstellt und auch verlangt.

- **Client-Programmierung**: Um eine effektive Client-Programmierung bereitzustellen, ist es mit dem .NET Framework möglich, sich einen Proxy für ein entferntes Service zu generieren. Die SOAP-spezifischen

Kommunikationsdetails, die notwendig für das Erstellen eines Funktionsaufrufs sind, werden für den Entwickler in einer sogenannten Stub-Klasse gekapselt, die ihrerseits die Methoden des entfernten Services zur Verfügung stellt, diese aber nicht selber implementiert hat.

Bei der Generierung eines solchen Proxys wird die Beschreibung des entfernten Services mittels WSDL zu Rate gezogen. Eine Hauptanwendung für diesen Zweck stellt der Befehl WSDL.EXE dar. Es ist möglich, aus einer WSDL-Beschreibung einen Proxy für eine beliebige Programmiersprache des .NET Frameworks zu erzeugen.

- **Implementierungsmodelle**: Das .NET Framework bietet zwei grundverschiedene Modelle, die Kommunikation mittels SOAP Nachrichten erlauben. Diese beiden Modelle wären zum einen der SOAP Stack des ASP.NET Pakets (vergleiche Abschnitt 3.3.2) und zum anderen die .NET Remoting Architektur (vergleiche Abschnitt 3.3.3).

- **Unterstützte Transportprotokolle**: In der aktuellen Version des .NET Frameworks wird für das ASP.NET Paket nur HTTP als Transportmittel für SOAP Nachrichten angeboten. Mittels .NET Remoting ist es hingegen aber durchaus möglich, auch TCP als Transportprotokoll zu verwenden. Bei ASP.NET kann TCP nur benutzt werden, wenn an beiden Enden der Kommunikation das Web Service Enhancement Paket installiert wurde.

3.3.6 Zukünftige Entwicklungen

In einer zukünftigen Version 2.0 des .NET Frameworks wird SOAP Version 1.2 unterstützt werden. Dies gilt in gut informierten Kreisen als sehr sicher. Vorabversionen der SOAP Spezifikation wurden schon in den Beta Versionen von .NET 1.1 eingebaut, aber kurz vor Herausbringen eines stabilen Produkts wieder aus diesem entfernt.
Bezüglich SOAP Nachrichten mit Attachments ist auch eine Erweiterung für die nächste .NET Generation geplant. Wie bereits in einem vorherigen Abschnitt berichtet, ist es momentan nur mit einem installierten Web Service Enhancement Paket möglich, SOAP mit Attachments zu verschicken. Es ist nötig, dass sowohl die Clientseite als auch die Serverseite über diese Erweiterung des .NET Frameworks verfügen. Hier sei auch noch erwähnt, dass unter der Federführung von Microsoft an einem neuen Standard für SOAP Nachrichten mit Attachments gearbeitet wurde, der über kurz oder lang der Nachfolger von MIME-kodierten Nachrichten sein soll. Im Moment wird in der ASP.NET Erweiterung noch die DIME-Kodierung für Nachrichten mit

Attachments verwendet. Diese soll aber bald durch die SOAP MTOM Spezifikation abgelöst werden ([57]).
Da Microsoft .NET bereits standardmäßig auf das Dokumentenaustauschmodell bei SOAP Nachrichten setzt, ist diese Unterstützung bereits in der jetzigen Version voll ausgeprägt. Da dieses Modell das Standardverfahren für mit .NET erstellte Services darstellt, kann es zu Interoperabilitätsproblemen mit anderen SOAP Kommunikationsplattformen kommen. Mehr zu diesem Thema aber in Abschnitt 4.2.2.
Natürlich gibt es auch bei Microsoft den Trend, die eigene SOAP Implementierung interoperabler zu gestalten. Aus diesem Zweck wurde von Microsoft ein eigener Server eingerichtet, der SOAP Dienste zur Verfügung stellt, mit denen man die Interoperabilität seiner eigenen Implementierung jener von Microsoft vergleichen kann. Problematisch ist dieser Bereich allemal, denn die Standardeinstellungen der .NET SOAP Implementierung korrespondieren nicht immer mit den Standardwerten der SOAP Spezifikation. Man darf hier vor allem in Hinblick auf die SOAP 1.2 Umsetzung gespannt sein, denn laut Spezifikation muss ja eine Implementierung nicht unbedingt das SOAP Encoding unterstützen, um als SOAP 1.2 konform zu gelten. Es wäre denkbar, dass im kommenden .NET Paket keine Unterstützung für das standardisierte SOAP Encoding vorhanden ist und nur mehr das dokumentbasierte Nachrichtenaustauschmodell verwendet wird.
Ein besonderes Augenmerk bei der Entwicklung einer zukünftigen SOAP Kommunikationsplattform unter .NET gilt der Sicherheit der Applikationen. Dieses Thema wurde in der Vergangenheit immer ausgespart und an den Rand gedrängt. Es stellt aber eine große Herausforderung für die Entwickler dar und ist in der Geschäftswelt ein nicht wegzudenkender Standard. Im Moment wird auf diesem Sektor sehr viel Arbeit geleistet, um die Versäumnisse der Vergangenheit nachzuholen ([38]). Eine dieser Initiativen ist der von Microsoft mitinitiierte WS-Security Standard. Mehr zu diesem Thema findet sich in [24]. In dieser Arbeit wird in Abschnitt 4.4 auf den Sicherheitsaspekt genauer eingegangen werden.

3.4 SOAP Kommunikation mit Suns JAX-RPC

Als nächste Implementierung wird Suns Java XML Remote Procedure Call Schnittstelle, nachfolgend kurz JAX-RPC genannt, näher untersucht. JAX-RPC ermöglicht die Ausführung RMI-artiger Remote Procedure Calls über SOAP ([50]).
Ziel der Implementierung soll es sein, die Umsetzung der SOAP Nachrichten und die SOAP Kommunikation für den Entwickler transparent zu halten,

indem dem Entwickler genau spezifizierte Schnittstellen zu den Kommunikationsmitteln zur Verfügung gestellt werden.

3.4.1 Überblick

JAX-RPC ist eine Spezifikation, die im Rahmen des Java Community Process (JCP) entwickelt wurde. An erster Stelle steht die Entwicklung eines JCP-genehmigten Standardsatzes von Java-APIs für das clientseitige als auch das serverseitige Programmiermodell. [50] bietet Einblick in die gesamte Spezifikation von JAX-RPC.
Die oben erwähnte Spezifikation umfasst folgende Kernstücke ([45]):

- Java Mapping: ein sehr umfassendes Type-Mapping System zwischen den Java Typen und den XML Schema Datentypen
- Service Endpoint: Modell eines Service Endpoints, Serviceprogrammierung
- Exception Handling: Fehlerbehandlung
- Service Kontext: Definition des Kontextes, Abarbeitung des Service Kontextes
- Message Handler: Handler APIs, Handler Modelle, Konfiguration von Handlern
- Service Clients: Clientprogrammierung mit JAX-RPC APIs
- „SOAP with Attachments“: Einbindung des SOAP Attachment Standards in die JAX-RPC Spezifikation
- Laufzeitumgebung: Definition der zur Verfügung gestellten Zusatzservices, Sicherheit, Session Management
- JAX-RPC Client Invocation Models

Auf einen Teil dieser verschiedenen von der Spezifikation genauestens behandelten Aspekte wird in den folgenden Abschnitten eingegangen werden.
Diese JAX-RPC APIs machen sich eine interoperable Kommunikation zwischen Java-Anwendungen durch ein Protokolldesign zunutze, dessen Kern auf SOAP basiert, aber nicht darauf beschränkt ist.
Die Spezifikation bietet nicht nur eine Beschreibung der Schnittstellen zu diesen APIs, sondern liefert auch die Vorlage zu einer Referenzimplementierung von Sun. Auf diese Referenzimplementierung wird in diesem Abschnitt

und auch im anschließenden Vergleich der Kommunikationsplattformen Bezug genommen. Diese SOAP Implementierung wird als Teil des Java Web Service Developer Pack (JWSDP) von Sun ausgeliefert. Dieses Paket bietet eine komplette Lösung in Java, die es ermöglicht, Services zu entwickeln und der Öffentlichkeit zur Verfügung zu stellen.

3.4.2 Umsetzung des Java Typsystems

Ein zentraler Punkt der JAX-RPC Spezifikation ist die Umsetzung der Java Standardtypen in ihr jeweiliges mehr oder weniger passendes Pendant in der XML Schema Definition. Es werden bereits in der Spezifikation gewisse Konvertierungsregeln festgelegt.
Tabelle 10 zeigt exemplarisch eine Auflistung der Java Datentypen mit ihren jeweiligen Entsprechungen, wie sie in der Spezifikation in [50] zu finden ist. Es sei nochmals darauf hingewiesen, dass diese Konvertierungstabelle bis auf wenige Ausnahmen mit der des Apache Axis Projekts identisch ist.
Zur Umsetzung der XML Datentypen bleibt hier noch anzumerken, dass laut Spezifikation des JAX-RPC eine konforme Implementierung den XML-Typ *xsd:anyType* nicht unterstützen muss. In der Referenzimplementierung werden aber sowohl der Typ *xsd:anyType* als auch *xsd:any* unterstützt. Diese Unterstützung lässt sich aber für die strikte Einhaltung der Spezifikation abschalten.
Die in Tabelle 10 angeführten simplen Datentypen dienen als Basis für die Umsetzung von komplexen Typen, wie es zum Beispiel ein Array darstellt. Anzumerken bleibt bei diesem Punkt, dass in der Spezifikation weitere XML Datentypen angeführt werden. Der größte Teil dieser zusätzlichen Typen wird aber auf den Java Typ `java.lang.String` abgebildet.
Eine Besonderheit, auf die in der Spezifikation zu JAX-RPC hingewiesen wird, ist, dass es nach der Spezifikation der SOAP Version 1.1 auch möglich ist, Elemente einer SOAP enkodierten Nachricht *nillable* zu setzen. Es ist nicht möglich, solche Werte auf einen primitiven Java Typ abzubilden. Hierbei ist es notwendig, die sogenannten Wrapper-Klassen für die primitiven Typen zu verwenden. Ein Beispiel hierfür wäre die Klasse `java.lang.Integer` für den primitiven Typ `int`.
Mittels des JAX-RPC API ist es möglich, Stubs aus WSDL-Beschreibungen zu generieren. Es wird von der Spezifikation sehr genau beschrieben, wie so ein Stub umzusetzen ist, denn es wurde für die Umsetzung solcher Objekte ein eigenes Interface eingeführt. Ebenso sind in der Spezifikation Templates für die Struktur und Nomenklatur solcher Stubs vorgesehen.
Ein weiterer wichtiger Punkt der Spezifikation ist die Generator-unterstützte Umsetzung von Java Service Klassen in eine WSDL-Beschreibung der öf-

XML Datentyp	Java Datentyp
xsd:string	java.lang.String
xsd:integer	java.math.BigInteger
xsd:int	int
xsd:long	long
xsd:short	short
xsd:decimal	java.math.BigDecimal
xsd:float	float
xsd:double	double
xsd:boolean	boolean
xsd:byte	byte
xsd:unsignedInt	long
xsd:unsignedShort	int
xsd:unsignedByte	short
xsd:QName	javax.xml.namespace.QName
xsd:dateTime	java.util.Calendar
xsd:date	java.util.Calendar
xsd:time	java.util.Calendar
xsd:anyURI	java.netURI oder java.lang.String
xsd:base64Binary	byte[]
xsd:hexBinary	byte[]
xsd:anySimpleType	java.lang.String

Tabelle 10: XML Datentypen und ihre Entsprechungen in Java laut der Spezifikation des JAX-RPC API

fentlichen Schnittstellen der entsprechenden Klassen. Auch hier wird von der Spezifikation genau vorgegeben, wie eine solche Umsetzung auszusehen hat. Es sollte aber bei der Implementierung, die die Spezifikation umsetzt, darauf geachtet werden, dass der Entwickler trotzdem noch Möglichkeiten der Einflussnahme auf die Umwandlung in die WSDL-Beschreibung hat.

3.4.3 Architekturelle Umsetzung

Einen Einblick in die serverseitige Architektur des SOAP Nachrichtenwegs zeigt die Abbildung 11. Hier wird der Weg einer SOAP Nachricht im Servlet Container, der den entfernten Service hostet, nachgezeichnet. An dieser Abbildung erkennt man schon, dass die Konzeption des Apache Axis Projekts sehr stark an die architekturellen Vorgaben der JAX-RPC Spezifikation angelehnt ist.
Zentrales Element bei der Abarbeitung einer SOAP Nachricht ist der Hand-

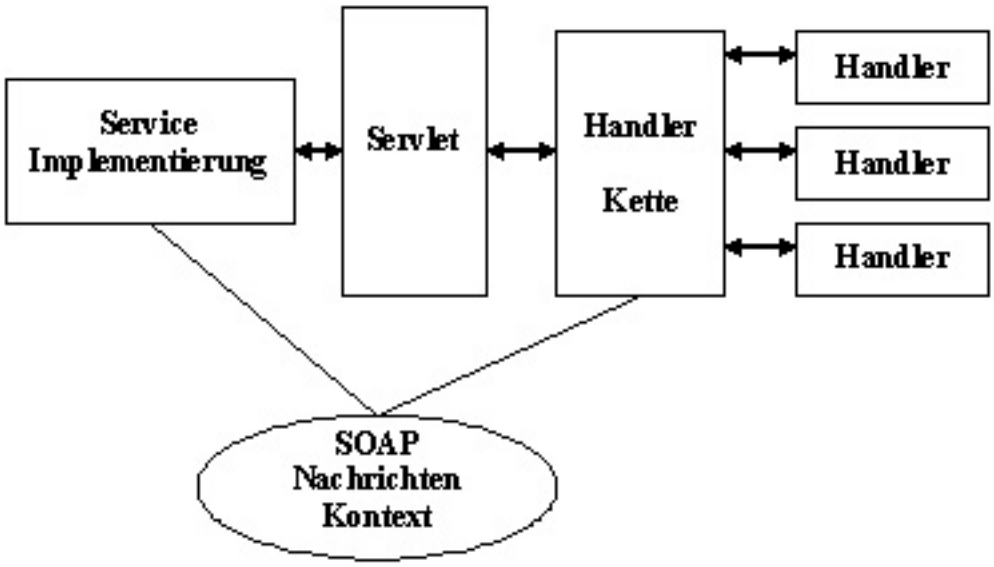

Abbildung 11: Serverseitiger Aufbau eines SOAP Services mit JAX-RPC

ler, dieser kann auch eine sogenannte Handler Chain sein, also eine Verkettung von mehreren Handlern. Handler können verschiedenste Aufgaben am Server übernehmen. Es steht ihnen jederzeit der SOAP Message Kontext für die Bearbeitung der Nachricht zur Verfügung. Eine mögliche Aufgabe eines Handlers wäre das Verschlüsseln beziehungsweise Entschlüsseln von Nachrichten, das Loggen von Nachrichten oder das intelligente Caching von Nachrichten beziehungsweise Antworten auf Anfragen.
Grundsätzlich sind Handler zustandslose Instanzen, die über die Laufzeitumgebung mit den notwendigen Kontextinformationen versorgt werden. Aus diesem Grund ist es einem Handler auch möglich, Zugang zu containerspezifischen Features zu bekommen.
In der Spezifikation zu JAX-RPC ist lediglich der SOAP Message Handler spezifiziert worden. SOAP Message Handler sind laut Spezifikation an Service Endpunkte an der Client- beziehungsweise Serverseite gebunden und erlauben dadurch zusätzliche Möglichkeiten, um eine SOAP Nachricht zu bearbeiten. Zum Beispiel wird die Bearbeitung von SOAP Headern oft in Handlern erledigt, die für das ausführende Service dann transparent Funktionalitäten übernehmen.

3.4.4 Der Weg einer SOAP Nachricht

Um den Weg einer SOAP Nachricht zu skizzieren, die mittels JAX-RPC erzeugt beziehungsweise empfangen wird, muss näher auf die Struktur eingegangen werden, die es erlaubt, Services für die Öffentlichkeit erreichbar zu machen.
Üblicherweise wird die JAX-RPC Laufzeitumgebung, die für die SOAP Kommunikation zwischen Client und Server zuständig ist, in einem Servlet Container eingebettet. Dieser Servlet Container übernimmt dann die HTTP Kommunikation und übergibt die empfangenen SOAP Nachrichten weiter an den SOAP Protokoll Handler. Dieser erhält die Möglichkeit, die empfangene Nachricht zu deserialisieren und in ein für Java Applikationen lesbares Format zu bringen. Die Umsetzung der XML Datentypen in Java Datentypen erfolgt wie in Abschnitt 3.4.2 beschrieben.
Dieser Prozess wird in JAX-RPC mittels einer Tie-Klasse gekapselt. Im Falle eines entfernten Prozeduraufrufs bedeutet das, die eigentliche Methode mit den erwarteten Parametern und Datentypen aufzurufen und dann die Rückgabewerte ebenfalls entsprechend verpackt an den Absender zurückzuschicken.

3.4.5 Leistungsmerkmale

In diesem Abschnitt sollen die einzelnen charakteristischen Merkmale dieser SOAP Implementierung näher beleuchtet werden.
Folgende Eigenschaften fallen besonders ins Auge:

- **Protokoll-Bindung**: Eigentlich ist die JAX-RPC Spezifikation für mehrere zu unterstützende Protokolle ausgelegt. Im Moment ist aber nur SOAP in der Version 1.1 umgesetzt worden. Eine Umsetzung des SOAP 1.2 Standards ist für eine zukünftige Version des JAX-RPC vorgesehen ([50]).

- **unterstützte Transportprotokolle**: Momentan wird HTTP in der Version 1.1 als Transportmittel für SOAP Nachrichten eingesetzt. Auch hier sei erwähnt, dass JAX-RPC nicht auf HTTP beschränkt ist. Die Hauptkomponenten des APIs sind Transportprotokoll neutral aufgebaut und könnten mit jedem beliebigen Protokoll kombiniert werden. Die Komponenten haben eben auch ein definiertes Protokoll Binding für SOAP/1.1 Nachrichten.

  Es bleibt anzumerken, dass mit JAX-RPC keine wie immer gearteten Sicherheitsmechanismen in Bezug auf das Transportprotokoll definiert

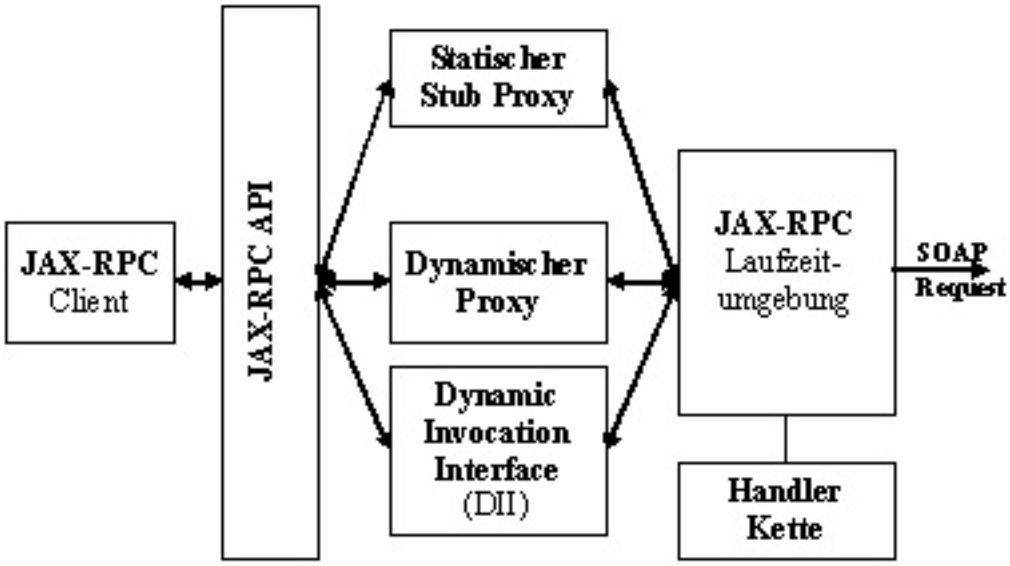

Abbildung 12: Die drei Möglichkeiten eines clientseitigen Serviceaufrufs mit JAX-RPC

wurden. Dies schließt aber nicht aus, dass Sicherheitsmaßnahmen bei Implementierungen der Schnittstellen eingefügt werden.

- **Implementierungen der Spezifikation**: Im Moment gibt es eine Referenzimplementierung von Sun, die auf der JAX-RPC Spezifikation aufbaut. Ebenso verwendet das Apache Axis Projekt, man vergleiche dazu Abschnitt 3.1, die Schnittstellendefinition von JAX-RPC und baut darauf die eigene SOAP Implementierung auf. Axis verwendet als Grundlage die Version 1.0 von JAX-RPC. Es ist Voraussetzung, dass eine Implementierung des JAX-RPC Laufzeitsystems den SOAP Standard „SOAP 1.1 with Attachments“ unterstützen muss. Zusätzlich gibt es noch einige kommerzielle Produkte, die ebenfalls auf die JAX-RPC Spezifikation aufbauen.

- **Kodierungseinstellungen**: Mittels JAX-RPC ist es möglich, sowohl kodierte Repräsentationen von SOAP Nachrichten zu verschicken beziehungsweise zu empfangen, als auch literale SOAP Nachrichten zu versenden beziehungsweise zu erhalten. Es wird für die literale Kodierung sowohl der dokumentbasierte Ansatz als auch der RPC-Stil unterstützt. Eine Unterstützung der literalen Kodierung ist auch zwingend für die WS-I Konformität der JAX-RPC Spezifikation notwendig gewesen.

Auch hier gilt wieder, dass es mittels JAX-RPC möglich ist, eigene Kodierungen einzubringen. Man muss sich immer bewusst sein, dass dies die Interoperabilität mit anderen Implementierungen empfindlich stört.

- **WSDL Umsetzung**: Es ist möglich, mittels der WSDL-Beschreibung eines Services eine solche Beschreibung in eine Java Schnittstelle umzuwandeln. Ebenso ist der umgekehrte Weg möglich, nämlich dass man eine Java-Schnittstelle in eine WSDL-Beschreibung für dieses Java Service umwandelt. Dies erfolgt bei der Sun Referenzimplementierung aber nicht zur Laufzeit des Services, wie es etwa bei Apache Axis der Fall ist, sondern die WSDL-Beschreibung des Services wird vorab von einem mitgelieferten Werkzeug erstellt.

- **umgesetzte SOAP Kommunikationsmodelle**: In der Spezifikation fand sowohl das Modell des synchronen Request-Response Einzug, als auch das Modell des Einweg-RPC seine Umsetzung. Es ist besonders beim synchronen Request-Response Modell anzumerken, dass hier der Rückgabewert auch vom Typ `void` sein kann.

- **SOAP Attachments**: JAX-RPC unterstützt den von Sun geförderten „SOAP with Attachment API for Java“ (kurz SAAJ genannt). Es ist also möglich, SOAP Nachrichten mit Attachments über MIME-kodierte HTTP Nachrichten zu verschicken.

- **Parameterübergabe**: Mittels JAX-RPC ist es lediglich möglich, Parameter als Kopien („by value“) zu übergeben. Eine Umsetzung eines „Object-by-reference Mode“ ist nicht vorgesehen. Ein interessantes Detail bezüglich dieser Java Implementierung ist die Tatsache, dass bei SOAP auch sogenannte Ausgabeparameter beziehungsweise Ein-Ausgabeparameter vorgesehen sind. Diese Parameter werden rein zur Rückgabe von Werten bzw. während der Abarbeitung der Funktion verändert. Solche Parameter werden in JAX-RPC durch Holder-Klassen implementiert.

- **Service Kontext**: Mittels JAX-RPC kann auch auf den sogenannten Service Kontext einer konkreten Applikation Einfluss genommen werden. Hinter diesem Service Kontext stehen eigentlich die SOAP Header.

  In diesem Zusammenhang sei noch einmal darauf hingewiesen, dass JAX-RPC nicht als Designziel hatte, Transaktionen oder Sicherheitskonzepte, die auf SOAP Header Informationen aufbauen, zu konzipieren.

- **Clientprogrammierung**: Bei JAX-RPC gibt ist es drei unterschiedliche Arten, ein entferntes Service aufzurufen. Zum einen wäre hier die Stubgenerierung zu nennen. Mit diesem Proxy kann man das Service direkt ansprechen. Die Umsetzung des Proxys erfolgt hier statisch, denn es wurde mittels einem WSDL2Java-Werkzeug eine statische Proxy-Klasse für den entfernten Dienst angelegt.

  Eine weitere Möglichkeit stellt der dynamische Aufruf mittels dem Service-Interface dar. Mittels dem dynamischen Proxy-Ansatzes kann ohne generierten Code, ein entferntes Service aufgerufen werden. Für die Ausführung eines entfernten Methodenaufrufs sind lediglich der namensraumqualifizierte Name des Service Ports und eine kompilierte Interface-Definitionsklasse notwendig.

  Als dritte Alternative eines Service Aufrufs sei hier das Dynamic Invocation Interface (DII) näher erklärt. Dieses DII basiert auf einem `Call` Objekt, das üblicherweise von einem `Service`-Objekt erzeugt wurde. Dieses `Call`-Objekt bietet verschiedenste Methoden um den Serviceaufruf zu initialisieren und schlussendlich zu starten. Interessant ist hier noch zu erwähnen, dass der Aufruf der Methode `invoke()` des `Call`-Objekts ein Objekt mit dem Rückgabewert der entfernten Servicemethode liefert. Weitere Ausgabeparameter lassen sich dann mittels einer eigenen Methode des `Call`-Objekts aus der Antwortnachricht extrahieren.

  Eine graphische Darstellung dieser drei Umsetzungen eines Serviceaufrufs findet sich in Abbildung 12.

- **Hosting**: Entwickler können ein Service mittels der Referenzimplementierung von JAX-RPC der Öffentlichkeit zur Verfügung stellen, indem die Implementierung in einem Java Servlet Container untergebracht wird. Standard ist, wie schon beim Apache Axis Projekt, der Jakarta Tomcat Servlet Container. Die Kombinierung mit eigenen Produkten von Sun wird aber empfohlen, da diese Systeme performanter sind.

- **Sicherheit**: Da die Spezifikation zu JAX-RPC sich eindeutig dazu bekennt, dass Security in Bezug auf SOAP Nachricht außerhalb dieser Spezifikation liegt, wurde in der Referenzimplementierung von Sun Unterstützung für SSL (Kurzbezeichnung für Secure Socket Layer) basierte Authentifizierung eingebaut. Mittels dieser Funktionalität ist es möglich, verschlüsselt Authentifizierungsdaten zu verschicken. Ebenso wird ein nicht standardkonformes API zur Signierung und Verifizierung von SOAP Nachrichten zur Verfügung gestellt. Leider handelt es sich um

ein nicht standardkonformes API, das in einer späteren Version sicher ersetzt werden wird.

3.4.6 Zukünftige Entwicklungen

Die JAX-RPC Spezifikation 2.0 wird der nächste große Meilenstein in der Entwicklung einer überarbeiteten Sun Referenzimplementierung sein. Darin soll dann auch schon die SOAP 1.2 Spezifikation ihre Umsetzung finden. Da man aber den Status der WS-I Konformität nicht verlieren möchte, wird sich die SOAP/1.1 Version als Standardeinstellung für Nachrichten mit der neuen Sun Referenzimplementierungsversion wiederfinden.
Ebenso ist eine Unterstützung von asynchronen SOAP Services geplant. Eine weitere Neuerung wird die Erweiterung des JAX-RPC APIs auf weitere Transportprotokolle sein. Im Moment ist es ja auf HTTP beschränkt. Man darf also gespannt sein, welche Neuerungen sich in der kommenden Version durchsetzen werden ([50]).

4 Der Vergleich

Dieses Kapitel ist dem Vergleich der eben näher vorgestellten SOAP Kommunikationsplattformen gewidmet. Dieser Vergleich ist aber nicht nur auf diese drei Implementierungen beschränkt, sondern soll einen umfassenden Überblick über die aktuelle Umsetzung von SOAP Plattformen bilden. Es ist mir auch wichtig, allgemeine Aussagen zu den untersuchten Thematiken zu treffen.
Ein erster Punkt, den es zu untersuchen gilt, ist der Stand der Umsetzung der SOAP 1.2 Spezifikation, die die Einleitung für diese Arbeit bildet.
Ein Schwerpunkt dieses Abschnittes wird die Interoperabilität darstellen. In diesem Punkt sollen die Schwachstellen des SOAP Protokolls, wie es im Moment von den Implementierungen genutzt wird, aufgezeigt werden. Dieser Abschnitt ist eher weniger auf die drei untersuchten SOAP Kommunikationsplattformen beschränkt, sondern soll auch einen Überblick darüber liefern, was zu tun ist, um ein möglichst interoperables Service zur Verfügung zu stellen.
Ein nicht minder wichtiger Teil dieses Abschnitts ist den SOAP Features gewidmet. In diesem Zusammenhang gilt es zu untersuchen, welche Erweiterungen der Basisfunktionalitäten einer SOAP Implementierung ihre Umsetzungen in den untersuchten Kommunikationsplattformen gefunden haben. Es gilt diese Features zu identifizieren, die vermeintlich den gleichen Zweck beziehungsweise die gleiche Aufgabe erfüllen sollen. Eine Untersuchung der zugrunde liegenden Technologien ist unumgänglich.
Ein abschließender Teil dieses Kapitels ist den sicherheitsrelevanten Funktionalitäten der SOAP Implementierungen gewidmet.

4.1 Umsetzung der SOAP Version 1.2

Fest steht, dass SOAP 1.2 in keiner der getesteten Implementierungen wirklich Einzug gehalten hat. Lediglich Apache Axis bietet Unterstützung für die Version 1.2. Es zeigt sich aber, dass es sich bei den verwendeten Namensräumen nicht um die der finalen SOAP 1.2 Spezifikation handelt. Vielmehr wurden hier die Namensräume einer Zwischenversion berücksichtigt. Bei der Untersuchung der Nachfolgeversion für die momentan als stabil gekennzeichnete Apache Axis Version stellte sich heraus, dass bereits die Namensräume der finalen SOAP 1.2 Spezifikation verwendet werden. Außerdem wird diese Version auch mit einer besseren Umsetzung der SOAP 1.2 Spezifikation aufwarten können. Apache 1.2 wird in der stabilen Version SOAP 1.2-konform sein.
Bei Suns JAX-RPC und bei Microsofts .NET Framework muss man schon

auf die nächsten Versionen warten, um SOAP 1.2-konforme Nachrichten verschicken zu können.
Als Resümee zu diesem Teilgebiet der Arbeit könnte man sagen, dass es sich bei SOAP 1.2 um eine teilweise erheblich überarbeitete Version des Standards handelt, die es ermöglicht, die Interoperabilität zwischen den einzelnen SOAP-Implementierungen zu erhöhen. Umsetzung fand die neue Spezifikation in den hier vorgestellten und oft verwendeten SOAP Kommunikationsplattformen aber nicht. Es gibt zwar schon erste Testimplementierungen der einzelnen Hersteller, aber die Werkzeuge, mit denen diese Testservices umgesetzt wurden, sind der Öffentlichkeit teilweise noch nicht zugänglich. Auch eine Anfrage meinerseits an Microsoft bezüglich deren Umsetzung der SOAP 1.2 Implementierung blieb unbeantwortet. Es bleibt also abzuwarten, wann sich SOAP 1.2 endgültig durchsetzen wird und die Nachfolge von SOAP 1.1 antritt, das ja allerorten als der aktuelle Standard gilt, den eine SOAP-Implementierung zu erfüllen hat. Im Moment ist die Unterstützung der neuen Protokollspezifikation aber eher mangelhaft bis gar nicht vorhanden. Dieses Ergebnis ist eigentlich enttäuschend, da der SOAP Version 1.2-Standard zum Zeitpunkt der Erstellung dieser Arbeit schon über ein Jahr alt ist.
Bezeichnend ist aber auch der Umstand, dass bei den untersuchten Implementierungen lediglich das Open Source Projekt Apache Axis bereits einen Teil der SOAP 1.2 Spezifikation umgesetzt hat. Dies zeigt wieder, wie die Implementierung einer neuen Technologie in der Open Source-Gemeinde unterstützt wird. Die Unterstützung erfolgt rasch, da versucht wird, die neuen Konzepte innerhalb kürzester Zeit in die Tat umzusetzen und somit einen wertvollen Einblick in die Möglichkeiten der neuen Technologie bietet. Deshalb ist es innerhalb einer relaziv kurzen Zeitspanne möglich, neue Versionen bewährter Software zu erstellen, die von einer Vielzahl von Entwicklern unterstützt beziehungsweise vorangetrieben wird.
Die mehr oder weniger proprietären Produkte von Microsoft und Sun haben noch keinen Schritt in Richtung SOAP Version 1.2 in ihren als stabil gekennzeichneten Versionen gewagt. Es bleibt also abzuwarten, wann diese Unternehmen nachziehen.

4.2 Interoperabilität zwischen den Implementierungen

Interoperabilität ist eines der großen Anliegen bei der Weiterentwicklung der SOAP Version zur jetzigen Version 1.2. Durch diesen Umstand sollte ein schon einige Jahre dauernder Prozess der Standardisierung weiter vorangetrieben werden.
Hier einige grundlegende Gedanken wie Interoperabilität zwischen einzelnen Implementierungen erreicht werden könnte, beziehungsweise welche grundle-

genden Regeln bezüglich der Interoperabilität zu beachten sind ([17]):

- **Verwendung der gleichen beziehungsweise einer kompatiblen Version der Spezifikation von SOAP zur Erzeugung beziehungsweise Bearbeitung einer SOAP Nachricht**: Wie bereits in Abschnitt 2.12 angedeutet, gibt es zwischen der SOAP Version 1.2 und 1.1 bedeutende Unterschiede, die eine nicht vollständige Abwärtskompatibilität verursachen.

- **Verwendung der gleichen Version von Services**: Auch Services, die über das Netz angeboten werden, unterliegen wie andere Software auch der stetigen Änderung durch die Entwickler. Auch hier kann es bei verschiedenen Versionen zu Konflikten kommen. Es kann passieren, dass neuere Versionen freigegeben werden, die dann mit den älteren Versionen nicht mehr kompatibel sind. Dies stellt ein großes Problem dar, da es für Versionierung von Services keine Standards gibt. Die Lösung dieses Problems liegt allein bei dem Provider des Services. Das Problem wirkt sich aber auch auf die Clientseite aus. Es gibt im Moment keine Möglichkeit, wie eine Clientapplikation herausfinden soll, welche Version eines Services angeboten wird. In diesem Bereich herrscht noch Handlungsbedarf, denn es fehlt an den nötigen Standards, um eine ordentliche Versionierung von entfernten Diensten zu ermöglichen.

Über diese allgemeinen Überlegungen hinaus kann die Interoperabilität an Interpretationen oder Missverständnissen der Spezifikationen, an der Unterstützung oder eben Nicht-Unterstützung optionaler Features in einem Service Standard, am Erweitern des Services mittels proprietärer Standards und Features oder eben, wie bereits beim Thema „Versionierung“ gesehen, an einem Fehlen von notwendigen Standards scheitern. Auf diese Aspekte soll in den folgenden Abschnitten näher eingegangen werden.

4.2.1 SOAP/1.1 und Interoperabilität

Die ersten SOAP-Interoperabilitätsprobleme ergeben sich hauptsächlich aus Mehrdeutigkeiten in der Spezifikation und den daraus resultierenden unterschiedlichen Interpretationen. Auch Implementierungen, die der Spezifikation vollständig gehorchen, können trotzdem noch zueinander inkompatibel sein. Im April des Jahres 2000 stellte Tony Hong von XMethods eine detaillierte Liste von Interoperabilitätsproblemen zwischen den einzelnen bekannten Implementierungen zusammen. Diese Liste zeigte, wie umfassend die Probleme damals waren ([17]).

Seit damals hat sich in diesem Bereich aber einiges getan, aber dazu in Abschnitt 4.2.7 mehr.
In den Entwürfen der Spezifikation für die Version 1.2 beschäftigte sich die XMLP mit den in der Version 1.1 gefundenen Interoperabilitätsproblemen, sodass in SOAP 1.2 viele der Probleme entweder ganz verschwunden oder wenigstens weniger problematisch sein sollten. Die XMLP hat außerdem zur verbindlichen Spezifikation noch ein nicht normatives Lehrbuch herausgegeben (siehe [51]), das einem die Verwendung von SOAP 1.2 näher bringen soll und auch eine Auflistung von Unterschieden zwischen der Version 1.1 und der Version 1.2 beinhaltet. Nebenbei wurde auch an einer Test-Suite zur Konformität gearbeitet. Das erwähnte Lehrbuch zu SOAP 1.2 und die Test-Suite sollen dazu beitragen, die Verwirrung und die Interpretationsunterschiede weiter zu reduzieren. Da aber SOAP/1.1 die derzeit am weitesten genutzte Form ist und sich auch in den in den vorangegangenen Abschnitten vorgestellten Implementierungen wiederfindet, ist es wichtig, sich die Bereiche dieser Spezifikation zu vergegenwärtigen, bei denen Interoperabilitätsprobleme zu vermuten sind.

4.2.2 Probleme durch die Spezifikation zu SOAP/1.1

In diesem Abschnitt sollen die Bereiche der SOAP 1.1 Spezifikation dargestellt werden, die ein Problem für die Interoperabilität darstellen können.

4.2.2.1 Kodierung

Wie bereits in Abschnitt 2.14 erwähnt, gibt es zwei grundverschiedene Modelle, wie SOAP genutzt werden kann. Diese beiden unterschiedlichen Modelle seien hier mit dem RPC-Modell und dem Dokumentaustauschmodell nochmals erwähnt. Die Kodierung und die Weise, wie diese beiden Modelle verwendet werden, sind eine der größten Herausforderungen, wenn es um Interoperabilität geht.
In SOAP 1.1 verlangt das RPC-Modell. Die Abbildung zwischen den XML Typen und den verschiedensten Typsystemen der Programmiersprachen ist eine sehr große Herausforderung für die Entwickler von SOAP Implementierungen. In diesem Zusammenhang gilt es festzuhalten, dass sich nicht alle Programmiersprachentypen gleich gut dafür eignen, in den jeweiligen XML-Typ umgewandelt zu werden. In diesem Zusammenhang sei noch die Typsicherheit als eines der größten Probleme in diesem Bereich erwähnt.
Da die SOAP Spezifikation sehr offen gestaltet wurde, gibt es auch keine Standard-Kodierung. Es wird zwar in der Spezifikation zu SOAP Version 1.1 ein Kodierungsstil definiert, der im Fachjargon auch Section 5 Encoding

genannt wird, weil er im Abschnitt 5 der SOAP Spezifikation näher beschrieben wird. Die Verwendung dieser Kodierung in einem Service ist aber nicht verpflichtend.
Weiters lässt sich der Kodierungsstil einer SOAP Nachricht nach der SOAP Spezifikation 1.1 auch noch über das Attribut *encodingStyle* festlegen. Die Funktion dieses Attributs wurde schon in Abschnitt 2.4.1 erläutert.
Zusätzlich zur RPC-Kodierung der Daten gibt es in SOAP 1.1 auch noch den Stil der literalen Kodierung von Daten, wie er schon in Abschnitt 2.14 näher beschrieben wurde.
Der Unterschied zwischen literaler und kodierter SOAP-Bindung stellt ein weiteres großes Problem der Interoperabilität dar. Dies geht vor allem darauf zurück, dass sich die verschiedenen Implementierungen nicht auf eine Standardkodierung von SOAP Nachrichten einigen konnten. Bei den drei vorher erwähnten SOAP Kommunikationsplattformen sind Apache Axis und die JAX-RPC Referenzimplementierung standardmäßig auf das Produzieren von RPC SOAP Nachrichten mit Section 5-Kodierung beziehungsweise literaler Kodierung eingestellt, während bei der ASP.NET Implementierung von SOAP der dokumentbasierte Ansatz mit literaler Kodierung verfolgt wird. Die Standardkodierung einer der erwähnten SOAP Implementierungen muss geändert werden. Bei .NET kann man SOAP kodierte Nachrichten mittels des Attributs `SoapRpcService` in der das Service implementierenden Klasse festlegen. Bei Apache Axis wird die literale Kodierung dadurch festgelegt, dass der Rückgabewert einer Servicemethode den Typ `org.xml.Document` hat. Um auch das dokumentbasierte Nachrichtenmodell zu unterstützen, dürfte die Servicemethode nur einen Parameter des Typs `org.xml.Document` haben. Ähnlich verhält es sich auch bei der JAX-RPC Referenzimplementierung. Kann nämlich das übermittelte XML-Dokument nicht einem Typ beziehungsweise einer Klassenrepräsentation zugeordnet werden, so wird das SOAP Body Element als `SOAPElement`-Objekt behandelt. Diese Objektklasse kann sowohl als Parameter als auch Rückgabewert einer Methode eingesetzt werden. Üblicherweise verlangt das Dokumentenaustauschmodell im Gegensatz zum Section 5-kodierten RPC-Modell keine verpflichtende Kodierung. Mit einer XML Schema-Empfehlung kann auch ein natives XML-Typsystem verwendet werden, das die Section 5-Kodierung oder eine andere spezialisierte Kodierung hier überflüssig macht.

4.2.2.2 Das SOAPAction HTTP Header Feld

Die SOAP 1.1 Spezifikation lässt die Verwendung des SOAPAction Elements offen für Interpretationen. Dies kommt dadurch zum Ausdruck, dass es bei HTTP als Transportprotokoll für SOAP Nachrichten als URI definiert ist,

der die, wie in der SOAP Spezifikation mehrdeutig formuliert, „Absicht“ der Nachricht angeben soll. Mit dieser Definition ist natürlich der Spekulation Tür und Tor geöffnet. Es könnte sich also beim Wert des SOAPAction Elements um das Ziel der Nachricht oder etwa um den Namen des Dienstes handeln, für den die Nachricht bestimmt ist. Es gibt aber noch durchaus andere Versionen, die das SOAPAction Element als Inidikator dafür nutzt, um zu bestimmen, welche Version eines Dienstes ausgeführt werden soll. Das Element soll somit bei der Versionierung von Services hilfreich sein. Dieser Umstand macht es durchaus einsichtig, dass das SOAPAction Element ein größeres Problem für die Interoperabilität von SOAP Plattformen darstellt, die mittels HTTP miteinander kommunizieren. Tatsache bleibt aber, dass es im Moment mehrere unterschiedliche Auffassungen gibt, wie das SOAPAction Element genutzt werden soll und welche Daten dieses Element mit sich führt.
In der Spezifikation zu SOAP 1.1 ist das SOAPAction Element auch für kein anderes Transportmittel außer HTTP definiert worden. Das Bereitstellen einer äquivalenten Funktionalität durch andere Transportprotokolle ist Sache der jeweiligen Implementierung. Dies gilt unabhängig davon, wie der Wert im SOAPAction Element von der Implementierung ausgelegt wird. Als Faustregel lässt sich formulieren, dass bei einer Verwendung eines anderen Transportprotokolls voraussichtlich Interoperabilitätsprobleme auftreten werden, bis Standards entwickelt werden, die genau spezifizieren, wie eine Bindung an andere Protokolle auszusehen hat.
Es soll nicht der Eindruck entstehen, dass das SOAPAction Element nur zur Minimierung der Interoperabilität vorhanden ist. Dies ist durchaus nicht der Fall. Die angebotene Funktionalität dieses Elements kann auch sehr nützlich sein. Es bietet nämlich die Möglichkeit, ankommende SOAP Nachrichten ohne das mühselige Parsen von XML auf den richtigen Weg zu bringen oder an eine bestimmte Adresse weiterzuleiten. Die oben skizzierten Probleme liegen viel mehr bei der Implementierung des SOAPAction Elements und der schwammigen Formulierung in der Spezifikation.
Wird nur HTTP als Transportprotokoll zwischen den SOAP Knoten verwendet, so kann es noch zusätzliche Verwirrung darüber geben, wie das SOAPAction Element interpretiert werden soll. In der Spezifikation zu SOAP 1.1 heißt es nämlich, dass das SOAPAction-Header-Feld den Firewalls die Möglichkeit geben soll, nach SOAP Nachrichten zu filtern. Wird in diesem Header-Feld ein leerer String als Wert erkannt, dann bedeutet das, dass der HTTP-Anfrage-URI die Absicht der SOAP-Nachricht mitteilt. Kein Wert im Header-Feld bedeutet, dass das SOAPAction Feld im HTTP-Header die Absicht der SOAP Nachricht nicht mitteilt. Somit kann man nach dieser Definition über zwei verschiedene Möglichkeiten, die Absicht einer Nachricht

bekanntgeben. Dies wäre zum einen über die URI und zum anderen über den SOAPAction Header. Problematisch ist aber das Nichtsetzen des Headers. Es ist nämlich nicht unbedingt einsichtig, ob es sich um einen NULL-String handelt oder ob eine leere Zeichenkette angegeben wurde. Dies kann für manche Server, die SOAP Services hosten, zu Problemen führen. Es ergibt sich in puncto Interoperabilität ein sehr großes Problem, wenn man von Implementierungsseite her auf dieses SOAPAction Element vertraut.
Nach einer langen und intensiven Debatte innerhalb der XMLP-Gruppe hat man in der Spezifikation zu SOAP Version 1.2 festgehalten, dass die Verwendung des SOAP Action-Features optional ist ([17]). SOAP-Empfänger können es als Hinweis zur Optimierung der Verarbeitung der empfangenen SOAP Nachricht einsetzen. Sie sollten sich aber nicht darauf verlassen, dass dieses Element immer anwesend ist. In der Spezifikation wurde auch die Unterstützung für das SOAPAction Element durch die Implementierungen als optional gekennzeichnet. Es wurde festgehalten, dass SOAP Implementierungen kein SOAPAction Element mehr erzeugen oder verlangen sollten, außer es gibt einen triftigen Grund, der zur Verwendung dieses Elements zwingt ([17]).
Die Formulierung in der Spezifikation garantiert auch keine vollständige Interoperabilität zwischen SOAP Kommunikationsplattformen ohne vorhergehende Absprache, verbessert aber die jetzige Situation doch um Einiges.
Beim ASP.NET Framework ist der SOAPAction Header unbedingt erforderlich. Standardmäßig gibt es nämlich die Servicemethode an, für die die Daten in der SOAP Nachricht bestimmt sind. Dieses Verhalten lässt sich zwar überschreiben, aber dieser Header ist in einer HTTP Anfrage an ein ASP.NET Web Service trotzdem notwendig. Es gibt im Moment keine Möglichkeit, auf den SOAPAction Header bei ASP.NET ganz zu verzichten.
Bei Apache Axis und der JAX-RPC Referenzimplementierung wird auf den SOAPAction Header nur zurückgegriffen, wenn dieser von der Serverimplementierung verlangt wird. Standardmäßig wird aber auf dieses Feld verzichtet und trägt an sich keine Daten, die zur Ausführung eines entfernten Dienstes notwendig sind. Hier ist man schon einen Schritt näher an der Spezifikation zu SOAP 1.2. Microsoft hat mit dem ASP.NET Paket diesen Schritt aber noch vor sich.

4.2.2.3 Verarbeitungsreihenfolge

Problematisch bei der Spezifikation des SOAP 1.1 Protokolls ist, dass keine echte Reihenfolge der Abarbeitung der einzelnen Elemente einer SOAP Nachricht angegeben wurde. Einer Implementierung, die die Spezifikation umsetzt, steht es also frei, in welcher Reihenfolge die SOAP Header und der

SOAP Body abgearbeitet werden. Demnach ist es auch möglich, dass während die Header bearbeitet werden, der Inhalt des SOAP Body Elements bereits ausgewertet wird.
In der SOAP 1.2 Spezifikation wurde den oben skizzierten Umständen Rechnung getragen, weil es dort einen Abschnitt gibt, der genau vorschreibt, wie eine SOAP Nachricht, die der Spezifikation entspricht, abgearbeitet werden muss. Die Verarbeitungsreihenfolge einer SOAP Nachricht in SOAP 1.2 wurde bereits in Abschnitt 2.5 näher erläutert.
Da die untersuchten SOAP Kommunikationsplattformen alle ausnahmslos SOAP/1.1 als Standard-SOAP-Version benutzen, liegt der Schluss nahe, dass es bei den verschiedenen Implementierungen einige Unterschiede in der Abarbeitungsreihenfolge der Elemente einer SOAP Nachricht gibt.
In Abschnitt 4.3.1 wird ein Problem bei der Abarbeitung von SOAP Nachrichten bei den untersuchten SOAP Implementierung näher erläutert.

4.2.2.4 SOAP Header und das *mustUnderstand*-Attribut

In der Spezifikation zu SOAP 1.1 wird ein Attribut namens *mustUnderstand* für SOAP Header Blöcke definiert. In dieser Spezifikation wird 0 für falsch beziehungsweise 1 für wahr angegeben. Mit der Spezifikation zu SOAP 1.2 wurden die möglichen Werte für dieses Attribut aber mit *true* und *false* festgesetzt. Es ergibt sich das Problem der Abwärtskompatibilität zwischen den einzelnen Versionen des SOAP Protokolls. Die Trennung der verschiedenen Versionen durch unterschiedliche Namensräume und den damit verbundenen unterschiedlichen Elementen in den Schemata dieser Namensräume sollten aber vor allzu ausufernden Interoperabilitätsproblemen schützen. Festgelegt ist in der Version 1.2 aber auch, dass Werte mit 0 oder 1 unterstützt werden sollten. Eine SOAP Kommunikationsplattformen, die also sowohl SOAP 1.2 als auch SOAP 1.1 unterstützt, sollte mit beiden Wertpaaren umgehen können.
Es hat einige Unterschiede in der Interpretation gegeben, wie ein Kommunikationsteilnehmer überprüfen soll, dass er alle SOAP Header Blöcke versteht, die mit einem *mustUnderstand*-Attribut mit dem Wert 1 oder *true* gekennzeichnet waren. Eine dieser Interpretationen besagt, dass es notwendig ist, alle Header, die das *mustUnderstand* Attribut gesetzt haben, vor der Verarbeitung zu prüfen ist. Eine andere Auslegung der Sachlage ist, dass jeder SOAP Header Block mit gesetztem *mustUnderstand* Attribut individuell geprüft und verarbeitet werden kann. Dies bringt naturgemäß mit sich, dass sich die Ergebnisse dieser beiden Arten der Verarbeitung sehr stark voneinander unterscheiden können.
Um die Verarbeitungsreihenfolge von für SOAP Header genauer festzulegen,

machte die XMLP konkrete Vorschläge, die sie in der Spezifikation zu SOAP Version 1.2 einfließen ließ. Die Reihenfolge der Verarbeitung von SOAP Headern wurde bereits in Abschnitt 2.5 dieser Arbeit näher erläutert.
Die Verarbeitung von SOAP Headern bei den drei untersuchten SOAP Kommunikationsplattformen wird in Abschnitt 4.3.1 genauer untersucht werden.

4.2.2.5 Serialisierung von programmiersprachenabhängigen Datentypen

In den meisten Implementierungen ist es möglich, dass man Datentypen, die nur in der jeweiligen Programmiersprache ihre Umsetzung finden, mittels XML serialisiert und dann mittels SOAP zwischen verschiedenen Anwendungen austauscht. Diese proprietären Datentypen stellen ein Problem für die Interoperabilität von SOAP Implementierungen dar, da sie nicht plattformübergreifend definiert sind und sich meist nur auf eine bestimmte Programmiersprache beschränken. Es kann durchaus möglich sein, dass nicht einmal Implementierungen, die sich auf das Typsystem derselben Programmiersprache stützen, die Daten richtig austauschen können.
Eine Implementierung, die nur die von XML Schema Spezifikation definierten Typen verwendet, hat sehr viel größere Chancen, interoperabel zu sein. Microsoft versucht mit der ASP.NET Implementierung der Web Service Komponenten diesen Weg zu gehen. Es wurde ja bereits im Abschnitt 3.3.2 gezeigt, dass hier vor allem auf Kompatibilität zu den XML Schema Datentypen wert gelegt wird. Dass dies nicht immer von Erfolg gekrönt ist, zeigt Abschnitt 4.2.5 dieser Arbeit.

4.2.2.6 Der Inhaltstyp bei SOAP Nachrichten

SOAP 1.1 spezifiziert den Inhaltstyp *text/xml* im HTTP-Header. Diese Typbezeichnung ist aber durchaus problematisch. Bei MIME-User-Agenten kann es möglich sein, das sie *text/xml* nicht explizit unterstützen, solche Dokumente als *text/plain* behandeln und den XML MIME-Part als reinen Text anzeigen.
Die XMLP-Gruppe hat sich deshalb dazu entschlossen, dass in der SOAP Version 1.2 nicht mehr der Typ *text/xml* verwendet wird, sondern stattdessen ein eigener MIME-Typ eingeführt wird. Dieser lautet *application/soap*, wobei *application* den MIME-Typ und *soap* den MIME-Subtyp angibt. Auch Erweiterungen dieses Schemas sind denkbar. Zum Beispiel könnte man eine XML serialisierte SOAP Nachricht der Version 1.2 mittels des Typen *application/soap+xml* kennzeichnen. Weil SOAP 1.2 auf dem XML-Infoset aufbaut, sind alternative Serialisierungen denkbar und auch zulässig.

Es steht fest, dass ab Version 1.2 von SOAP der Typ *text/xml* für SOAP Nachrichten nicht mehr unterstützt wird.
Apache Axis unterstützt die neuen Inhaltstypen. Konkret wird der Typ „*application/soap+xml*“ verwendet. Bei den beiden anderen SOAP Kommunikationsplattformen ist nach wie vor der „*text/xml*“ Inhaltstyp in Verwendung.

4.2.2.7 Ausführung von Services mit unvollständigen Parametern

Ein Problem, dass sich auch auf die SOAP Spezifikation zurückführen lässt, hängt mit der Ausführung von Servicemethoden zusammen, deren Parameter nicht vollständig angegeben wurden. Da es die SOAP Spezifikationen den Implementierungen freistellt, ob sie Methoden ausführen, denen Parameter fehlen, ist es oft nicht vorhersagbar, ob eine Servicemethode ohne Probleme ausgeführt werden kann. Dieser Bereich betrifft übrigens nur die „RPC/encoded“ Services, da bei diesen Parameter an die Methoden übergeben werden. Wird das dokumentbasierte Nachrichtenaustauschmodell verwendet, so sind die Nachrichten, die von einem Service verarbeitet werden können durch die XML Schema Beschreibung vorgegeben.
Bei der JAX-RPC Referenzimplementierung ist es zum Beispiel so gelöst, dass eine Methode ausgeführt wird, indem zu Anfang alle Parameter mit einem Standardwert initialisiert werden. Mittels dieser Implementierung können auch Methoden mit unvollständigen Parametern ausgeführt werden. Für Objekte wird hier typischerweise der Wert `null` angenommen. Die primitiven Java-Typen werden ebenfalls mit Standardwerten versehen. Es bleibt aber anzumerken, dass dies zu Nebeneffekten führen kann, die nicht vom Entwickler beabsichtigt wurden.
Bei Apache Axis ist es ebenfalls möglich, dass Servicemethoden ausgeführt werden, deren Parametersatz nicht komplett ist. Es kommt aber darauf an, ob der nicht übertragene Parameter ein Java Objekt oder Primitiv ist. Handelt es sich um ein Objekt, so wird der entsprechende Parameter einfach mit `null` initialisiert. Ist der Parameter aber ein Java Primitiv, so wird ein SOAP Fault ausgelöst, weil keine Operation gefunden werden konnte, die der Signatur in der SOAP Nachricht entsprochen hätte.
Somit ergibt sich ein vollkommen unterschiedliches Verhalten der Apache Axis Implementierung für ein und den selben Parameter eines Services. Für die Clientapplikation ist es nämlich aus dem WSDL nicht ersichtlich, ob der Parameter durch ein Java Primitiv oder durch eine etwaige Wrapper-Klasse in die Signatur der Methode eingeflossen ist. Die Darstellung eines solchen Parameters ist nämlich identisch. Um eine Servicemethode auch mit unvollständigen Parametern ausführen zu können, ist es unbedingt notwendig, dass

man Wrapper-Klassen als Parameter für solche Methoden verwendet anstatt Java Primitive.
Wie bei der Sun Referenzimplementierung ist es auch mittels ASP.NET möglich, entfernte Dienste mit unvollständigen Parametern aufzurufen. Da es sich bei allen CLR-Typen um Objekte handelt, ist dies einfach zu bewerkstelligen, indem alle nicht übertragenen Parameter den Wert `null` zugewiesen bekommen. Wie auch bei der JAX-RPC Referenzimplementierung gilt es hier etwaige unerwünschte Nebeneffekte zu beachten.

4.2.3 XML Schema

Zur Zeit gibt es drei Versionen des XML Schema Definition-Namensraums (XSD), die in den Jahren 1999, 2000 und 2001 standardisiert wurden. Die Verwendung verschiedener XML Schema-Versionen kann zu Namensraumproblemen oder Problemen mit der Serialisierung und Deserialisierung führen. In der letzten Version der XML Schema Definition haben mehrere Datentypen andere Namen bekommen. Hier sei beispielhaft der Typ *xsd:dateTime* genannt, der in den Vorversionen *timeInstant* genannt wurde.
Die meisten Werkzeuge sind dahingehend verändert worden, dass sie mit der Version aus 2001 klarkommen sollten. Diese stellt nämlich im Moment so etwas wie den Standard der unterstützten XML Datentypen dar.
Von den drei untersuchten SOAP Implementierungen unterstützen alle die XML Schema Spezifikation aus dem Jahr 2001. Es sollte von der XML Schema Unterstützung her keine Probleme geben. Zu untersuchen gilt es in diesem Punkt aber noch, wie gut beziehungsweise wie schlecht die XML Datentypen auf das Typsystem der jeweiligen Programmiersprachen umgesetzt wurden und welchen Einschränkungen diese Umsetzungen unterliegen. Eine Untersuchung dieses Umstandes findet man in Abschnitt 4.2.5 dieser Arbeit.

4.2.4 WSDL und Interoperabilität

Im Moment gibt es zwei Lager mit deutlich unterschiedlichen Ansichten über den Nutzen und Wert von WSDL in Bezug auf SOAP Services. Im einen Lager stehen so anerkannte Persönlichkeiten wie Dave Winer von Userland, die glauben, dass WSDL nur ein Interoperabilitätshindernis ist. Sie meinen, dass WSDL einfach zu kompliziert ist und weniger offen als zum Beispiel HTTP, XML oder auch SOAP. Der WSDL Standard würde zu Bindungen an gewisse Hersteller einladen. Weiters behauptet dieses Lager, dass die Folge der Anwendung von WSDL kleinere Marktchancen für kleine, unabhängige Akteure sei ([17]).

Die andere Partei in dieser Meinungsverschiedenheit besteht aus großen Firmen wie etwa Microsoft und IBM. Diese behaupten, dass WSDL die Interoperabilität und die Entwicklung und Implementation von Services über Plattformen und Werkzeuge hinweg durch standardisierte Definition und Dokumentation der Dienste verbessere.
Fest steht, dass WSDL für die Interoperabilität von Services nicht unbedingt erforderlich ist. Sie dient vielfach dazu, um dem Entwickler das Leben zu erleichtern. Mit einer Schnittstellendefinition durch eine WSDL-Datei, ist es bei vielen Implementierungen möglich, sich ein Proxy-Objekt für das entfernte Service zu erzeugen, das dann die Kommunikation zwischen der lokalen Anwendung und dem entfernten Service kapselt. Es ist möglich, sehr einfach und auch dynamisch Services in eigene Anwendungen einzubinden. Dies kann in manchen Szenarien nützlich sein, ist aber, wie gesagt, nicht zwingend erforderlich. Die Daten über die Schnittstelle eines Services können auch ohne WSDL Beschreibung zwischen Entwicklern ausgetauscht werden.
Bezüglich den drei untersuchten SOAP Kommunikationsplattformen kann man in diesem Punkt feststellen, dass sich alle drei Implementierungen stark auf den WSDL-Standard stützen. Dies sieht man vor allem daran, dass es in jeder der drei Plattformen Werkzeuge gibt, mit denen man WSDL Beschreibungen für eigene Services generieren kann, beziehungsweise Proxy Services mittels eigenen Tools erstellt.
In der jetzigen Version ist es mittels Apache Axis auch möglich, dass man mittels `Call`-Objekt ohne einen statisch oder dynamisch generierten WSDL-Proxy entfernte Services aufrufen kann. Dieses `Call`-Objekt stammt im Übrigen aus der JAX-RPC Spezifikation und steht auch der Sun Referenzimplementierung zur Verfügung. Mehr zu dieser Art der Clientprogrammierung findet sich in Abschnitt 3.4.5.
Das .NET Framework von Microsoft verlässt sich nur auf den WSDL Standard bei der Generierung und Ausführung von entfernten Services. Dies war in einer der Vorgängerversionen zur jetzigen Version 1.1 ein Problem, da die Datentypen in den SOAP Nachrichten, die von Services, die mit dem .NET Framework erzeugt wurden, keine Typinformation in der SOAP Nachricht enthielten, die angeben konnte, welcher Typ das serialisierte XML Element darstellte. Die Typinformation konnte nämlich nur aus der WSDL-Beschreibung des Services entnommen werden. Frühere Apache SOAP Implementierungen hatten ein großes Problem mit diesem Verhalten, da sie ihrerseits keine WSDL unterstützten und somit die SOAP Nachrichten von den beschriebenen .NET Services nicht deserialisieren konnten.
Mittlerweile gehört die Unterstützung von WSDL-Beschreibungen und die automatische Service-Proxy-Generierung schon zur Standardausstattung einer SOAP Kommunikationsplattform. Ausnahmen bilden hier die SOAP Im-

plementierungen einiger Skriptsprachen, die im Moment nur experimentelle Unterstützung für WSDL-Generierung beziehungsweise Verarbeitung bieten.

4.2.5 Unterschiedliche Umsetzung von XML Schema Datentypen

In diesem Abschnitt soll näher beleuchtet werden, wie sich die Umsetzung der einzelnen XML Schema Datentypen auf das jeweilige Programmiersprachen Typsystem, das den einzelnen SOAP Kommunikationsplattformen zu Grunde liegt, auf die Interoperabilität der einzelnen SOAP Implementierungen auswirkt. Hier wird aber auch berücksichtigt, wie gut sich der jeweilige Programmiersprachentyp auf den XML Schema Datentyp umsetzen lässt und umgekehrt.
Wie schon Sam Ruby in seinem berühmten Artikel in [58] feststellt, ist die Interoperabilität von Implementierungen immer auch sehr eng mit der Umsetzung des Typsystems in XML verknüpft. Die XML Schema Spezifikation definiert in [12] sehr genau, welche Anforderungen der XML Datentyp bezüglich Wertebereich und ähnlichem eigentlich zu erfüllen hat. Dies korrespondiert aber oftmals nicht mit der Datenstruktur der Programmiersprache, welche die im XML der SOAP Nachricht versandten Werte speichern soll. Ziel dieses Abschnitts ist es, solche Datentypen bei den drei vorgestellten SOAP Kommunikationsplattformen zu identifizieren und eventuelle Probleme in deren Umsetzung aufzuzeigen. Es ist festzustellen, dass sich die meisten Differenzen auf Unterschiede zwischen den Java Datentypen und den .NET CLR Datentypen beziehen. Da sich die Apache Axis SOAP Implementierung ja auf die JAX-RPC Spezifikation stützt und die darin definierte Umsetzung der Datentypen 1:1 übernommen hat, ergeben sich bei diesen beiden Implementierungen auf den ersten Blick keine Unterschiede in der Umsetzung der Datentypen. Auf den zweiten Blick dann schon, aber dazu später mehr.
Ein Datentyp, der sich bei näherer Betrachtung als problematisch herausstellt, ist der Typ *xsd:decimal*. Definiert ist dieser Typ als Dezimalzahl mit beliebiger Genauigkeit. Eine Minimalanforderung der XML Schema Spezifikation ist, dass eine Implementierung mindestens 18 Stellen verarbeiten können muss. Es wird aber darauf hingewiesen, dass dies eine applikationsabhängige Limitierung der Unterstützung dieses Datentyps bedeutet und die maximale Anzahl an Stellen von Applikationsseite her klar dokumentiert werden muss. Dieser Punkt ist im Bereich von Web Services als eher problematisch zu sehen. Denn es liegt im Bereich des Diensteanbieters zu entscheiden, was mit Zahlen passieren soll, die die nötige Genauigkeit vermissen lassen, beziehungsweise wie viel Genauigkeit von den Client-Applikationen erwartet wird.
Nun betrachten wir die Umsetzung des *xsd:decimal* Datentyp in Apache Axis

und JAX-RPC. Man sieht in Tabelle 7 und Tabelle 10 sofort, dass dieser Typ auf den Java Datentyp `java.math.BigDecimal` abgebildet wird. Dieser Typ verfügt über eine Genauigkeit von mehreren Milliarden Stellen und übertrifft somit bei Weitem den in der XML Schema Spezifikation verlangten Wert von 18 Stellen Genauigkeit ([12]). Mit dem Datentyp `java.math.BigDecimal` ist es leicht möglich, dass man Berechnungen mit fast beliebiger Genauigkeit durchführt. Hier wird der Exponent und die Mantisse getrennt gespeichert. Sendet eine Applikation Daten mit mehreren 100 Stellen Genauigkeit, so wird diese Zahl bis auf die letzte Stelle genau von den Java SOAP Implementierungen gespeichert. Wie schon beim vorherigen Beispiel liegt es dann wieder im Bereich der Applikation zu entscheiden, ob ein eventueller Verlust von Genauigkeit schwerwiegende Folgen hat. Von den Java Umsetzungen von SOAP wird auf jeden Fall eine sehr hohe Genauigkeit unterstützt.
Problematischer wird dieser Datentyp bei Microsofts .NET Framework. Dort wird der *xsd:decimal* Datentyp, wie in Tabelle 8 bereits angedeutet, auf `System.Decimal` abgebildet. Dieser Typ bietet eine Genauigkeit von 25 Stellen. Diese liegt zwar über der Mindestanforderung der XML Schema Spezifikation, aber weit hinter der Genauigkeit in den Java Implementierungen zurück.
Wie man sieht, lassen sich zwischen den Java SOAP Implementierungen und der .NET SOAP Kommunikationsplattform nur Dezimalzahlen bis zu einer Genauigkeit von 25 Stellen ohne Präzisionsverlust übertragen. Zwischen den beiden Java SOAP Implementierungen ist ein Austausch von Zahlen mit fast beliebiger Genauigkeit möglich.
Ein weiterer XML Datentyp, der sehr problematisch sein kann, ist der Typ *xsd:dateTime*, der dazu verwendet wird, um Zeitstempel zu übermitteln. In der XML Schema Spezifikation wird dieser Typ als zusammengesetzte Zeichenkette definiert, die aus jeweils zwei Ziffern für Jahrhundert, Jahr, Monat, Tag, Stunden, Minuten und Sekunden definiert wird. Für eine feinere Unterteilung der Zeitmarken ist aber von der Spezifikation her noch vorgesehen, dass man die Sekunde als Bruchteil darstellen kann. Diese Bruchteile einer Sekunde können dann beliebig genau gewählt werden. Eine Einschränkung der Genauigkeit ist hier nicht getroffen worden, was die Lage zusehends kompliziert. Es ist somit nur gesichert, dass ein Zeitstempel bis zur Sekunde genau ist. Eine genauere Zerlegung der Zeit ist nicht mit absoluter Sicherheit von einer Applikation zu erwarten.
In Apache Axis und der JAX-RPC Spezifikation wird der XML-Datentyp *xsd:dateTime* auf den Java Typ `java.util.Calendar` abgebildet. Dieser Typ an sich unterstützt Zeitwerte bis hin zur Millisekunde. Eine weitere Unterteilung des Zeitstempels ist aber nicht mehr möglich. Anders ist der Umstand beim .NET Framework von Microsoft. Dort ist es ohne Probleme möglich,

bis zu 100 Nanosekunden-Einheiten genau einen Zeitwert zu bestimmen. Dies ergibt eine Differenz von 4 Stellen bei der Genauigkeit.
Die Problematik ergibt sich auch hier wieder beim Austausch der Zeitwerte zwischen den verschiedenen Implementierungen. Es liegt an der SOAP Serializer beziehungsweise Deserializer Einheit, in welcher Form die übermittelten Daten schlussendlich in die Applikation einfließen. Eine Vorhersage, wie die Werte umgesetzt werden, ist oftmals kaum möglich, denn die Daten werden zwar von Java-Seite her validiert und verarbeitet, aber die Präzision der Werte vernachlässigt. Es ist nicht mehr genau feststellbar, ob der übermittelte Wert auch dem entspricht, was die Applikation erhält. Dies ist wieder ein Beispiel, wie sich plattformspezifische Einschränkungen auf die Interoperabilität zwischen SOAP Implementierungen auswirken. Eine Auswertung der Genauigkeit eines Zeitstempels zwischen .NET und Java ist bis zu einem Millisekunden-Wert noch akzeptabel, wenn man andere SOAP Implementierungen betrachtet, die die Verarbeitung des XML Typs *xsd:dateTime* schon nach den Sekundenwerten einstellen.
Ein Datentyp, der bereits in der XML Schema Spezifikation mehrdeutig definiert wurde, ist *xsd:boolean*. Dieser Typ darf sowohl die Werte *true* oder *false* als auch „1“ beziehungsweise „0“ annehmen. Es ergibt sich die Problematik, dass eine SOAP Implementierung, die die XML Schema Spezifikation voll unterstützt, mit beiden möglichen Wertepaaren umgehen können muss. Die Standardumsetzung dieses Typs erfolgt aber auf die Werte `true` beziehungsweise `false`. Apache Axis unterstützt bei der Deserialisierung beide Wertpaare. Darüber hinaus wäre es auch möglich, dass man ein drittes Paar benutzt, das aus den Werten `True` beziehungsweise `False` besteht. Dies ist durch einen Blick in den Source-Code des Projekts einfach festzustellen gewesen. Ein interessantes Detail am Rande ist, dass bei der Deserialisierung des Typs `boolean` nur der erste Buchstabe des Wertes verglichen wird, somit wären noch ganz andere Wertpaare möglich. Für die Serialisierung dieses simplen Datentyps wird die `toString()`-Methode des Objekts aufgerufen. Fakt ist aber, dass das Apache Axis Projekt auch SOAP Nachrichten annimmt, die nicht standardkonform sind, ja sogar als fehlerhaft bezeichnet werden können.
Bezüglich der .NET Umsetzung von boolschen Werten ist zu sagen, dass C#, das auf dem .NET Framework aufbaut, kein „0“ oder „1“ als Wert für boolsche Variablen zulässt. Eine Untersuchung der Umsetzung dieses Typs ist also notwendig. Es ist von besonderem Interesse, wie die .NET SOAP Implementierung mit dem boolschen Wertpaar „0“ und „1“ umgeht, dass von der XML Schema Spezifikation als möglicher Wert für ein XML Element vom Typ *xsd:boolean* erlaubt ist. Um Probleme mit diesem Datentyp zu entgehen, wird im `XmlSerializer` eine Funktion `convertBoolean()` aufgerufen, die

die nötigen Konvertierungen übernimmt und im Fehlerfall einen SOAP Fault generiert. Somit ist es in der SOAP Implementierung von ASP.NET auch möglich, das Wertpaar 0 und 1 für boolsche Werte zu benutzen.
Ein generelles Problem für die Interoperabilität von SOAP Implementierungen stellen primitive Datentypen wie `unsigned Integer`, `unsigned Long` oder ähnliche dar. Diese Datentypen werden von Suns Java nicht unterstützt. Ein spezieller Typ ist da auch noch der Character-Datentyp (`char`), denn dieser Typ wird in der XML Schema Spezifikation nicht erwähnt, was die ganze Problematik zusätzlich verschärft, denn es ist den Implementierungen freigestellt, wie sie diesen Datentyp in einer SOAP Nachricht umsetzen. Die nicht vorzeichenbehafteten Typen sind in der XML Schema Spezifikation aber sehr wohl erwähnt, was in einigen Fällen auch nicht wirklich weiterhilft.
Eine Verwendung dieser Datentypen führt mit ziemlicher Sicherheit zu Interoperabilitätsproblemen aufgrund plattformspezifischer Unterschiede. Der problematische Aspekt ist, dass es mittels dem .NET Framework möglich ist, solche Datentypen als Parameter für eine Methode eines Services zu verwenden. Hiermit macht man das Service für Java SOAP Implementierungen aber fast unbenutzbar, denn es gibt in Java keinen standardisierten Weg, wie solche mittels SOAP verschickte Daten verarbeitet werden sollten.
Apache Axis hat im Gegensatz zu der JAX-RPC Referenzimplementierung auch Unterstützung für vorzeichenlose Datentypen integriert ([3]). Dies erlaubt es somit, auch mit anderen Programmiersprachen wie etwa C# oder C++ mittels dieser Datentypen zu kommunizieren, ohne dass diese Typen in Java integriert sind. Zu diesem Zweck wurden in der Apache Axis Bibliothek eigene Klassen für die jeweiligen Datentypen angelegt, die die Funktionalität von unsigned Datentypen übernehmen. Man handelt sich Probleme mit der JAX-RPC Referenzimplementierung von Sun ein, die diese Art von Datentypen überhaupt nicht unterstützt. Dies ist also ein Punkt, der die beiden Java SOAP Implementierungen in puncto Interoperabilität von einander unterscheidet. Der SOAP Stack des .NET Frameworks kennt keine Probleme mit den nicht vorzeichenbehafteten Datentypen der XML Schema Spezifikation.
Ein etwas komplexerer Datentyp des .NET Frameworks ist der Aufzählungstyp (`Enumeration`). Dieser Typ wird aber bei der Serialisierung als leerzeichengetrennte Liste im XML Element dargestellt. Dies ist eine sehr problematische Umsetzung, mit der die Java SOAP Implementierungen nicht klar kommen und diesen CLR-Typ als nicht für die Interoperabilität zwischen diesen Implementierungen förderlich kennzeichnen. Axis erkennt diese XML Schema Datenstruktur nicht richtig und kann daher den Aufzählungstyp nicht mehr rekonstruieren. Ähnliches gilt aber auf Java-Seite auch, wenn man versucht, eine `java.util.Hashtable` als Parameter einer Methode zu verschicken, denn dieser Datentyp wird vom .NET Framework nicht richtig

deserialisiert. Möglich ist der Austausch dieses Datentyps aber zwischen den meisten Java SOAP Implementierungen. Fast alle diese SOAP Kommunikationsplattformen unterstützen die komplexeren Typen `java.util.Hashtable` und `java.util.ArrayList`.

Datentyp	**Axis**	**Sun RI**	**.NET**
XML Schema Typ			
xsd:decimal	+	+	+/-[1]
xsd:dateTime	+/-[2]	+/-[2]	+
xsd:boolean	+	+	+
xsd:float	+	+	+
xsd:unsignedInt	+	-	+
xsd:unsignedLong	+	-	+
xsd:unsignedShort	+	-	+
xsd:unsignedByte	+	-	+
Sonstige Typen			
`Hashtable`[3]	+	+	-
`Enumeration`[4]	-	-	+

Tabelle 11: Umsetzung von Typen in den einzelnen Kommunikationsplattformen

Es wäre gezeigt, dass es nicht immer nur an der SOAP Spezifikation scheitern kann, sondern auch SOAP Nachrichten verschickt werden können, die gültige Werte enthalten, aber an der Empfängerseite durch plattformspezifische Einschränkungen zu Problemen mit der Nachricht führen, die man eigentlich nicht berücksichtigt hat. Die Problematik liegt hier beim Entwickler der Anwendung, der sich auf die SOAP Kommunikation stützt und versuchen muss, möglichst alle Client-Implementierungen zu unterstützen. Der Entwickler hat zu entscheiden, wie mit den fehlerhaften Daten umgegangen werden muss, beziehungsweise welche Fehlermechanismen ausgelöst werden müssen. Auf jeden Fall muss man sich dieses Sachverhalts jederzeit bewusst sein und solche Fehlerquellen in der Applikation abfangen. Von SOAP Seite her sind diese Interoperabilitätsprobleme nicht in den Griff zu bekommen, da sie sich auf fundamentale Unterschiede zwischen den Plattformen stützen. Dies ist eben der Preis, den SOAP zahlen muss, weil es als plattformübergreifendes Kommunikationsmittel konzipiert und verwendet wird.

[1] nur bis zu 25 Stellen Genauigkeit
[2] nur auf Millisekunde genau
[3] Java Typ
[4] .NET Typ

Ein interessanter Punkt zur Unterstützung von ungültigen Werten, damit ist ungültig im Sinne der XML Schema Definition gemeint, bei SOAP Nachrichten Elementen ist die Geschichte zur Serialisierung beziehungsweise Deserialisierung von `float`-Gleitkommazahlen in Apache Axis. Wie bereits in Abschnitt 3.1.1 erwähnt, stammt Axis ursprünglich von der IBM SOAP4J Implementierung ab. Diese serialisierte alle simplen Datentypen mittels der `toString()`-Methode von Java Objekten. Deserialisiert wurden diese Typen wieder mittels der String Konstruktoren der jeweiligen Wrapper-Klassen. Problematisch war aber der Wert der Unendlichkeit bei Gleitkommazahlen, der bei Java mit dem Literal `Infinity` bezeichnet wird. In der XML Schema Spezifikation wird aber die Unendlichkeit mittels der Zeichenkette *INF* gekennzeichnet. Da aber SOAP4J dieses Format nicht einlesen beziehungsweise liefern konnte, kam es zu sehr großen Problemen mit anderen SOAP Implementierungen. Die Nachfolgeversion von SOAP4J und gleichzeitig Vorgängerversion von Axis, Apache SOAP, konnte diesen Fehler aber nur teilweise ausmerzen. Man hatte zwar den Fehler bei der Deserialisierung behoben und konnte somit auch *INF* auf `Infinity` abbilden, aber eine Serialisierung einer unendlichen Gleitkommazahl lieferte noch immer `Infinity`, was noch immer zu Problemen mit anderen SOAP Implementierungen führte, aber auf Grund der Abwärtskompatibilität zu SOAP4J notwendig war. Erst Apache Axis konnte den Fehler vollends ausbessern, um den Preis, dass diese SOAP Implementierung nicht mehr problemlos interoperabel mit der Vorvorgängerversion SOAP4J war, aber damit ein Problem mit vielen anderen SOAP Implementierungen ausgeräumt hat. Als kleines Detail am Rande sei hier noch erwähnt, dass die Apache Axis Implementierung aber nach wie vor noch *xsd:float*-Datentypen verarbeiten kann, die als Wert *Infinity* stehen haben. Hier ist man der Urimplementierung treu geblieben.
Tabelle 11 zeigt zusammenfassend die in diesem Abschnitt erwähnten Typen und deren Umsetzung.

4.2.6 Weitere Interoperabilitätsprobleme

Um zum Abschluss dieses Abschnitts einen weitreichenderen Einblick in die Probleme mit der Interoperabilität zu bekommen, sollen in diesem Abschnitt Ansatzpunkte für Problemstellungen erläutert werden, die sich nicht direkt mit der Umsetzung dieser Problematiken in den SOAP Kommunikationsplattformen beschäftigen, sondern eher die Rahmenbedingungen für die Erstellung von spezifikationskonformen Implementierungen betreffen.
So gibt es generell ein Problem mit den Entscheidungsschichten. Es sind nämlich mehrere Erweiterungen für Themen wie etwa Sicherheit und Zuverlässigkeit notwendig, um die bisher vorhandenen Services aufzuwerten. Im Moment

fehlt aber noch eine Entscheidung der Standardisierungsgremien, in welche Gebiete diese Serviceerweiterungen eigentlich fallen. Als Beispiel wollen wir die Thematik „Zuverlässigkeit“ näher betrachten. Da gab es durch die Firma IBM einen Vorschlag für das sogenannte HTTPR, eine Kurzform für Reliable HTTP. Dies würde bedeuten, dass das Thema „Zuverlässigkeit von Services“ auf der Transportprotokollebene angesiedelt wäre ([23]). Man stieß aber bei anderen Organisation auf Widerwillen, denn es war unter diesen die Meinung vertreten, dass Zuverlässigkeit nicht zum Transportprotokoll gehöre. Deshalb gab es auch von ebXML die Initiative, Zuverlässigkeit zum Nachrichtenprotokollstandard hinzuzufügen ([54]). Ähnliche Überlegungen gab und gibt es auch für die Themen Sicherheit ([55]) und Dienstequalität (Quality of Service) ([47]).
Ein weiteres Problem ist die Entwicklung und der Wildwuchs von Standards. Das W3C und andere Standardisierungsgremien versuchen möglichst viele Lücken in den Spezifikationen zu ermitteln, um sie in neuen Versionen der Spezifikationen ausmerzen zu können. In diesem Zusammenhang stellen Zuverlässigkeit und Sicherheit von Services die Hauptthemen dar. Es gibt aber unter den diversen Gremien keinen Konsens, welche Standards am kritischsten sind. Problematischer ist der Umstand, dass sich die einzelnen Standardisierungsgremien untereinander nicht koordinieren, sodass bei vielen Themen mehrere konkurrierende Standards entstehen. Oft kommt es auch zu Überschneidungen der Themengebiete. Die Verwirrung um die Standards wird dadurch komplettiert, dass sich vertikale Märkte oftmals eigene Standards entwerfen, um den Austausch von Geschäftsdaten in oder zwischen diesen Märkten zu beschleunigen.
Eine Folge der oben genannten Entwicklungen ist, dass sich meistens der am häufigsten verwendete Standard durchsetzt, ungeachtet dem Umstand, dass es vielleicht andere Standards gibt, die das Problem besser lösen würden. Im Moment entwickelt sich Interoperabilität mehr und mehr zum Albtraum. Die derzeitigen SOAP Implementierungen müssen bezüglich verschiedenster Themen einen Mindeststandard anbieten. Diese Mindeststandards gehen aber oftmals weit über das hinaus, was die derzeitig allgegenwärtigen Standards wie SOAP, XML oder HTTP an zusätzlichen Features bieten können.
Einige Initiativen, die es sich zum Ziel gesetzt haben, die Interoperabilität zwischen den SOAP Implementierungen zu fördern, werden im nächsten Abschnitt näher vorgestellt.

Kriterium	**Axis**	**Sun RI**	**.NET**
Stil & Kodierung			
RPC/encoded	+[5]	+[5]	+

[5] Standardkodierung

Kriterium	Axis	Sun RI	.NET
RPC/literal	+/-	+	-
Document/literal	+/-	+	+[5]
SOAPAction notwendig	-	-	+
Verarbeitung von *mustUnderstand* Headern	+	+	+
Ausführung mit unvollständigen Parametern	+/-[6]	+	+
Unterstützung von XML Schema 2001	+	+	+
WSDL Unterstützung	+	+	+
Unterstützung der neuen Inhaltstypen	+	-	-

Tabelle 12: Gegenüberstellung einiger Interoperabilitätskriterien

Tabelle 12 zeigt eine Gegenüberstellung einiger in dieser Arbeit identifizierter Interoperabilitätskriterien und deren Umsetzung in den untersuchten SOAP Kommunikationsplattformen.

4.2.7 Lösungen zum Interoperabilitätsproblem

In den vorherigen Abschnitten wurden einige Interoperabilitätsprobleme zwischen SOAP Implementierungen aufgelistet. Es soll aber nicht der Eindruck entstehen, dass ständig mehr Probleme hinzukommen und sich niemand um die Lösung der anstehenden Probleme kümmert. Dies ist nämlich absolut nicht der Fall.

4.2.7.1 Die SOAPBuilders Interoperabilitätstests

Innerhalb der SOAP Entwicklergemeinde hat man schnell erkannt, dass Interoperabilität zwischen den einzelnen Implementierungen sehr wichtig ist. Ebenso ist es wichtig, sich über identifizierte Probleme auszutauschen und Lösungen für die anstehenden Interoperabilitätsprobleme zu finden. Aus diesem Grund wurde die SOAPBuilders-Gruppe ins Leben gerufen, die sich als Ziel gesetzt hat, die Interoperabilität zwischen den Implementierungen zu fördern. Im Moment handelt es sich bei der SOAPBuilders-Diskussionsgruppe um ein Forum für SOAP-Implementatoren, indem sich Entwickler untereinander mit Fragen der Interoperabilität und diversen Spezifikationen auseinandersetzen können. Die Gruppe existiert schon seit Jänner 2001 und umfasst im Moment mehr als 1400 Mitglieder. In den Anfangsjahren dieser Gruppe wurde dieses Forum von seinen Mitgliedern sehr aktiv genutzt. Jetzt dient es

[6] nur bei Verwendung von Wrapper-Klassen anstatt primitiver Java Typen

mehr zum Austausch von Problemen und Missverständnissen zwischen Applikationsentwicklern und den Entwicklern der SOAP Implementierungen.
Hauptziel der SOAPBuilders ist das Einrichten von verschiedenen Laboratorien für Interoperabilitätstests der SOAP 1.1 Implementierungen. Diese Testlabors stellen Test-Suiten bereit, die SOAP-Implementationen zum Testen ihrer Interoperabilität verwenden können. Historisch gesehen basieren die SOAPBuilders Tests auf Tests, die entwickelt wurden, um die Interoperabilität zwischen Apache SOAP und SOAP::Lite für die Skriptsprache Perl zu verbessern. Diese Tests stellen eine Sammlung von SOAP-RPC-Echo-Aufrufen dar. WSDL-Dokumente und Browser-basierte Clients zur Verwendung der Services zählen bei allen vorhandenen Test-Suiten zum Standard. Ein Verzeichnis von Endpunkten für jede Implementation ist im Laboratorium vorhanden ([17]).
Die erste Runde dieser Interoperabilitätstests wurde von XMethods organisiert und im Juni 2001 durchgeführt. Die nächste Runde (Round 2) der Interoperabilitätsaktivitäten der SOAPBuilders wurde im Jahr 2002 unter der Führung von Bob Cunnings von White Mesa lanciert. Für jede dieser Runden wurden Listen mit den teilnehmenden Implementierungen erstellt. Pro Implementierung wurden dann WSDL-Beschreibungen der implementierten Services erstellt und die Testergebnisse dokumentiert. Auf Grund dieser Ergebnisse war es möglich einzuschätzen, wie interoperabel die jeweilige SOAP Implementierung gerade war.
Nach der Vollendung der jeweiligen Testrunden wurden Updates für die meisten Implementierungen herausgebracht, um die festgestellten Mängel, Unzulänglichkeiten und Interoperabilitätsprobleme zu beheben. Viele der in der ersten Runde genannten Dienste existieren nicht mehr, weil auch die zugrunde liegenden SOAP-Implementierungen veraltet sind und nicht mehr dem neuesten Stand entsprechen.
Die angebotenen Interoperabilitätstests wurden allesamt für SOAP Version 1.1 angeboten und durchgeführt. Für SOAP Version 1.2 gibt es im Moment eigentlich keine offiziellen SOAPBuilders Tests. Es steht aber vom W3C eine Auflistung von Testimplementierungen verschiedener Hersteller zur Verfügung, die SOAP 1.2 unterstützen. Leider ist diese Auflistung doch etwas sehr veraltet und mangelhaft.

4.2.7.2 Web Service Interoperability Organization

Neben der bereits oben erwähnten SOAPBuilders-Gemeinde, die ihre Aktivitäten auf die Entwickler der einzelnen Implementierungen stützte, gibt es mittlerweile auch einen Zusammenschluss von Firmen, die SOAP Implementierungen zur Verfügung stellen. Unter der Schirmherrschaft großer Firmen

wie Microsoft, IBM oder Sun Microsystems entwickelte sich die Web Services Interoperability Organization (WS-I). Es handelt sich bei diesem Zusammenschluss aber nicht nur um die „Großen“ im Gewerbe, sondern es gibt auch eine Vielzahl von kleineren Unternehmen die aktiv am Diskussions- und Arbeitsprozess der Gruppe teilnehmen.
In diesem Zusammenhang ist auch zu bemerken, dass Sun Microsystems keines der Gründungsmitglieder dieser Organisation im Februar 2002 war. Später wurde nämlich bekannt, dass Microsoft sich nur dann bereit erklärt hatte, dieser Gruppe beizutreten, wenn Sun keiner der Hauptvertreter sein würde. Es handelte sich um eine rein politische Entscheidung. Später ist aber auch Sun der Vereinigung beigetreten. Im Zuge dieser Erweiterung hat sich dann auch der Einfluss von Sun auf die Organisation entsprechend vergrößert ([59]).
Bei WS-I handelt es sich um eine offene Industrieorganisation, die es sich zur Aufgabe gestellt hat, die Web Service Interoperabilität über Plattformen, Betriebssysteme und Programmiersprachen hinweg zu fördern. Diese Organisation besteht sowohl aus Mitgliedern aus der Industrie als auch aus Leuten, die in Standardisierungsorganisationen zu finden sind. Die Aufgaben der Mitglieder besteht darin, Kunden durch Anleitungen, Hilfestellungen sowie Ressourcen zur Entwicklung von möglichst interoperablen Web Service Lösungen zu verhelfen.
Bei WS-I handelt es sich um keine Standardisierungsorganisation, sondern um eine Institution, die es sich zur Aufgabe gestellt hat, Profile für bereits existierende Spezifikationen zu entwerfen. Es werden also keine neuen Technologien erfunden, sondern die vorhandenen Technologien auf ein möglichst interoperables Profil zusammengeschnitten.
Grundgedanke war, dass jeder Standard seine spezifischen Probleme hat, aber die Entwickler, welche Services auf Basis dieser Standards entwerfen, sich die Probleme aller Standards einhandeln können. Es gibt nämlich unendlich viele Interpretationen der Standards, die zu unendlich vielen verschiedenen Problemen für Entwickler führen können. Diesen Entwicklern soll mittels des Profils geholfen werden. Es auch zu berücksichtigen, dass die Konsumenten der Standards oftmals auch Plattformen von verschiedenen Herstellern verwenden, was die gesamte Sache zu einer großen Herausforderung für die Interoperabilität macht.
Aus diesen Gründen wurde auch im Jahr 2004 ein Basic Profile erstellt, das ein Profil beschreibt, welches angibt, wie die von einer dem Profil entsprechenden SOAP-Implementierung generierten SOAP Nachrichten auszusehen haben und welche SOAP Features anzubieten sind, um möglichst interoperabel zu sein. Es werden Regeln definiert, die von der SOAP Implementierung strikt einzuhalten sind, um die Profil-Konformität zu erlangen ([29]).

Das Basic Profile lässt sich in vier verschiedene Teile gliedern:

- **Nachrichtenbeschreibungen**: Es wird definiert, wie WSDL dazu benutzt werden kann, um das Format einer Nachricht zu beschreiben. Hier wird auf die WSDL Version 1.1 näher eingegangen.
- **Entdeckung**: Mit diesem Punkt ist gemeint, wie UDDI (Universal Description, Discovery and Integration) dazu eingesetzt werden kann, um das richtige Service in einer Sammlung von Services zu finden.
- **Nachrichten**: Dieser Punkt beinhaltet eine Beschreibung, wie SOAP und HTTP genutzt werden können, um servicespezifische Nachrichten auszutauschen.
- **Sicherheit**: In diesem Punkt geht es darum, wie HTTPS eingesetzt werden kann, um die Privacy und Integrität der ausgetauschten Nachrichten zu garantieren.

Zur obigen Aufzählung ist aber nochmals festzuhalten, dass es in diesem Basic Profile nicht darum geht, applikations- beziehungsweise geschäftsspezifische Prozesse und Abhängigkeiten abzubilden. Hier geht es viel mehr darum, dass grundlegende protokollspezifische Sachverhalte festgehalten werden, die es ermöglichen, dass Konsumenten und Services einfach und erfolgreich Nachrichten austauschen können. Vom Profil her ist es nicht vorgeschrieben, alle festgelegten Mechanismen für den Nachrichtenaustausch auch zu verwenden. Das Profil besagt nämlich nur, dass man, wenn man zum Beispiel SOAP als Nachrichtenformat verwendet, sich an die Richtlinien des Profils bei der Anwendung des Protokolls halten muss.
Interessant bei diesem Basic Profile ist, dass es für eine konforme Implementierung nicht erlaubt ist, „RPC/encoded“ Services anzubieten. Begründung dafür ist, dass das Section 5-Encoding ein unüberwindbares Hindernis für die Interoperabilität der Implementierungen darstellt. Diese Behauptung wurde nach Meinung der Expertenrunde des WS-I durch die Arbeit der SOAP-Builders Interoperability Gruppe untermauert. Diese hat sich bei ihren Tests auch sehr intensiv um RPC Services mit SOAP-Kodierung gekümmert und versucht dort die Interoperabilitätsprobleme zu lösen. Ein weiteres Argument für das Auflassen des SOAP-Encodings war die Spezifikation zu SOAP Version 1.2 selbst. Dort heißt es eben, dass eine Implementierung auch SOAP 1.2-konform sein kann, wenn sie die SOAP Kodierung von Funktionsaufrufen nicht unterstützt ([59]).
Schade ist, dass das Basic WS-I Profile auf SOAP/1.1 aufbaut. Eine Erweiterung für SOAP Version 1.2 ist wenn überhaupt in einem zukünftigen Profil

zu erwarten. Als Grund, warum das finale Basic Profil SOAP 1.1 vorschreibt, wird von WS-I-Seite her angegeben, dass es sich bei SOAP 1.1 um einen De facto-Standard handelte, als man mit der Erstellung des Profils begonnen hatte. Außerdem unterstützen die meisten aktuellen SOAP Implementierungen diese Version. Des weiteren will man noch abwarten, wie SOAP Version 1.2 von den Entwicklern angenommen wird und dann gegebenenfalls Schritte in diese Richtung unternehmen ([29]).
Als weiteren Standard, der Einfluss auf ein Profil hat, wurde die WSDL-Spezifikation gewählt. Es sollen Zweideutigkeiten in dieser Spezifikation auf eine eindeutige Vorgehensweise umgesetzt werden, die Interoperabilität garantiert.
Für das Testen, ob eine SOAP Implementierung konform zum Basic Profile ist, wurden von der WS-I Test-Tools entworfen. Dies soll sicherstellen, dass die Implementierung einen Mindestsatz an Anforderungen implementiert. Ebenso soll es Beratung für Benutzer von SOAP Kommunikationsplattformen geben, die dabei helfen soll, die richtige Implementierung für den jeweiligen Einsatzzweck zu finden. Als Maßstab hierbei dient die Interoperabilität der Implementierung.
Die im Moment bindende Version ist die Version 1.0 des Basic Profile ([31]). Zur Zeit der Erstellung dieser Arbeit wird aber schon an der nächste Version gearbeitet. Dieses Profil ist dann neu strukturiert, um eine bessere Komposition von Elementen des Profils zu gewährleisten. Außerdem werden Fehler der aktuellen Version in der kommenden ausgemerzt.
Als Ergänzung zum Basic Profile wird zur Zeit noch das Attachment Profile entworfen. In diesem Profil soll SOAP mit Attachments näher beschreiben werden. Ebenso hat dieses Profil auch Einfluss auf die WSDL Beschreibung. Es soll nämlich in der WSDL-Datei eines Services, das Attachments unterstützt, das MIME-Format der Nachrichten beschrieben werden ([32]). Dieses Profil stand zur Zeit der Erstellung dieser Arbeit knapp vor der Fertigstellung.
Als weitere Ergänzung zum Basis-Profil soll es ein Basic Security Profile geben. In diesem Profil sollen potentielle Sicherheitsrisiken und Bedrohungen von Services näher beleuchtet werden. Dazu zählen vor allem das Identifizieren und Authentifizieren der Gegenstelle, Non-Repudiation, Datenintegrität und Datengeheimhaltung. Es sollen Maßnahmen gefunden werden, die dann in das Profil einfließen sollen. Diese fallen in den Bereich von SSL/TLS, X509 Zertifizierung oder HTTP Basic Authentication. Eine erste Veröffentlichung dieses Profils wird noch für Ende des Jahres 2004 angestrebt ([8]).
Apache Axis hat in der aktuellen stabilen Version 1.1 das Basic Profile noch nicht umgesetzt. Da die Grundlagen dieses Profils aber auch in die Spezifikation von JAX-RPC Version 1.1 eingeflossen sind, ist eine Umsetzung des

Profils in der Apache Axis Version 1.2 geplant ([3]).
Die JAX-RPC Spezifikation 1.1 ist als WS-I konform gekennzeichnet. Wenn man noch die Unterstützung des SOAP Attachment Features mit MIME-Kodierung hinzurechnet, so ist die Sun Referenzimplementierung die einzige hier vorgestellte SOAP Kommunikationsplattform, die die von der WS-I vorgeschlagenen Profile vollkommen unterstützt.
Um zu überprüfen, ob ein erstelltes ASP.NET Service dem Basic Profile entspricht, kann man das von Microsoft entwickelte Test-Tool verwenden. Es ist dann auch möglich, dass man bereits existierende SOAP Dienste nach Profil-Konformität untersucht. Die Problematik der Versionierung gilt es aber zu beachten. Eine Umstellung eines bereits existierenden Services bedingt immer auch Änderungen in den Clientanwendungen. Einen guten Überblick über mögliche Stellen eines ASP.NET Services, die gegen die Vorgaben des Basic Profile verstoßen können, findet sich in [28].

4.2.7.3 Die interoperable Lösung

In den vorhergehenden Abschnitten wurden Initiativen genannt, mit denen Interoperabilitätsprobleme erkannt beziehungsweise ausgemerzt werden. In diesem Abschnitt soll eine zugegebenermaßen sehr restriktive Lösung präsentiert werden, die ein größtes Maß an Interoperabilität zwischen den einzelnen Implementierungen garantieren soll.
Wie bereits in Abschnitt 4.2.5 gezeigt wurde, gibt es eine erhebliche Anzahl von XML Datentypen, die durch ihre Umsetzung auf Typen der einzelnen Programmiersprachen Probleme zwischen den SOAP Implementierungen verursachen. Diese Probleme heißt es zu umgehen. Es sind für die Implementierung eines Services nur Datentypen zu verwenden, deren Umsetzung in die XML-Datentypen als sicher zwischen den Implementierungen zu betrachten ist. Da diese Menge von Typen recht begrenzt ist, soll sie hier aufgezählt werden. Die folgenden Typen lassen sich ohne Bedenken zwecks Interoperabilität in einem Service verwenden:

- String: Eine endliche Anzahl von Zeichen sollte von jeder Implementierung verarbeitet werden können.
- Integer: Eine ganze Zahl, die mittels 32 Bit dargestellt werden kann. Auch dieser Datentyp ist in jeder Implementierung vorhanden.
- Float: Gleitkommazahlen zählen bei den meisten Programmiersprachen zum Standard. Die Darstellung dieser Zahlen ist genormt, und die Verarbeitung sollte daher keine Probleme bereiten.

Es gilt zu beachten, dass es bei den meisten Datentypen keine Probleme gibt, solange man sich nicht in Randbereichen oder Sonderfällen dieses Typs

befindet. Man vergleiche dazu die bereits vorher erwähnten Probleme bei den XML Datentypen, die in Abschnitt 4.2.5 näher erläutert wurden.
Generell kann man sagen, dass Lösungen, die programmiersprachenabhängige Typen verwenden, ziemlich sicher Probleme bezüglich der Interoperabilität mit anderen SOAP Implementierungen bekommen, weil die Serialisierungs- beziehungsweise Deserialiseriungsinstanzen der verschiedenen SOAP Plattformen sehr wahrscheinlich nicht mit den Daten umgehen können. Deshalb sollte man im Sinne der Interoperabilität bei der Erstellung eines Services, der sich auf eine möglichst große Basis von Client-Implementierungen stützen will, auf Datentypen, die nur in der einen Programmiersprache vorkommen, verzichten.
Ein ebenso guter Tipp ist es, bei den Typen der verwendeten Parameter darauf zu achten, dass es sich nicht um zu komplexe Strukturen handelt. So gewagte Konstrukte wie teilweise übertragene Felder oder multidimensionale Arrays, wie sie in der SOAP Spezifikation 1.1 zu finden sind, sind im Einsatz für ein Service nicht geeignet, da es bei den meisten SOAP Implementierungen keine weitreichende Unterstützung für solche Daten gibt. Bei den meisten Implementierungen ist es maximal möglich, Felder von Feldern zu verschicken. Eine tiefere Verschachtelung kann dann zu unerwünschten Nebeneffekten führen, da diese Art der Kodierung von fast keiner Implementierung unterstützt wird. In der Apache Axis Anleitung wird sogar extra darauf hingewiesen, dass nur Arrays mit bis zu zwei Dimensionen verschickt werden können ([3]).
Beherzigt man diese Vorschläge, sollte es kein Problem darstellen, ein möglichst interoperables Service zu erstellen. Das dies aber keine Dauerlösung sein kann, liegt aber auf der Hand. In nächster Zeit wird sich alleine durch die Initiative des WS-I mit ihrem Basic Profile schon einiges auf diesem Gebiet tun. Erste Umsetzungen dieser Gedanken in verwendbaren Produkten lassen aber noch auf sich warten. Es wird seine Zeit dauern, bis dieses Profil allgemeine Akzeptanz und Umsetzung findet. Bis zu diesem Zeitpunkt wird sich die Interoperabilität zwischen den einzelnen Implementierungen aber mit größter Wahrscheinlichkeit nicht bessern. Dies liegt zum einen an den mehrdeutigen und ungenauen Spezifikationen beziehungsweise Standards, aber zu einem nicht geringen Teil an den Implementierungen selbst, die diese Spezifikationen an einigen Stellen nicht konform umgesetzt haben und somit Probleme bei deren Einsatz verursachen.

4.3 Zusätzliche Funktionen der SOAP Implementierungen

In diesem Abschnitt werden die einzelnen Leistungsmerkmale der in Abschnitt 3 vorgestellten Implementierungen nochmals genauer untersucht und miteinander verglichen. Interessant wird in diesem Kapitel sein, welche sogenannten SOAP Features ihre Umsetzung in den einzelnen Implementierungen gefunden haben, beziehungsweise ob und wie diese plattformunabhängig eingesetzt werden können.

4.3.1 Zugriff auf SOAP Header

SOAP Header sind ein elementares Element, wenn es um die Erweiterbarkeit von SOAP geht. Die meisten Erweiterungen werden mittels SOAP Header-Blöcken realisiert. Ein Beispiel, wie etwa ein SOAP Header Block zur Übertragung von sicherheitsrelevanten Daten benutzt werden kann, zeigt Abschnitt 4.4.
Diese Header stellen ein unverzichtbares Instrument für jeden Entwickler von entfernten Services dar. Es muss aber der Applikation die Möglichkeit gegeben werden, auf diese Daten innerhalb des Servicecodes zugreifen zu können. Dies wäre ein wichtiger Punkt für die Benutzbarkeit einer SOAP Kommunikationsplattform.
Mittels Apache Axis ist es relativ einfach, auf die Elemente eines SOAP Headers zuzugreifen. Auf Serviceseite gilt es zu beachten, dass das aktuelle `MessageContext` Objekt die notwendigen Daten enthält. Durch dieses Objekt erhält man Zugriff auf die aktuellen SOAP Header der empfangenen oder ausgehenden Nachricht.
Es erhebt sich die Frage, wie es mit Headern auf der Clientseite aussieht. Die Antwort lautet, dass es hier genau so einfach ist, Header zu setzen oder aus der Antwort zu extrahieren. Das zentrale Element hier ist das `Call`-Objekt das Zugriff auf die Header erlaubt. So ist auch bei clientseitigen Applikationen ein leichter Umgang mit SOAP Headern möglich.
Generell kann man sagen, dass bei Axis mittels dem `SOAPHeaderElement`-Objekt ein leichter Umgang mit SOAP Headern möglich ist. Dieses Objekt bietet die grundlegenden Funktionalitäten, um sinnvoll mit diesem SOAP Nachrichten Element arbeiten zu können. Der Einsatz von SOAP Headern ist mittels des Apache Axis Projekts uneingeschränkt möglich.
In interessantes Detail stellt die Behandlung von SOAP Headern mit gesetztem *mustUnderstand*-Attribut dar. Diese müssen ja von dem Dienst verstanden werden, was so viel bedeutet, dass der Dienst diese SOAP Header verarbeiten können muss. Signalisiert wird dieser Umstand im Apache

Axis Framework dadurch, dass man mittels der Methode `setProcessed()` der `SOAPHeaderElement`-Klasse das entsprechende Attribut auf wahr setzt. Durch den Aufruf dieser Methode ist der SOAP Header als verarbeitet markiert. Problematisch hierbei ist aber, dass dieses Attribut vom Entwickler selber gesetzt werden muss und dies nicht etwa beim Auslesen der SOAP Header gemacht wird. Weiters ist es ein Problem, dass erst nach Abarbeitung der Service-Methode überprüft wird, ob alle *mustUnderstand* Header auch verarbeitet worden sind. Da sich Apache Axis als SOAP 1.2 konform ausgibt, dürfte dies nicht der Fall sein, denn hier müssten alle *mustUnderstand* SOAP Header vor dem SOAP Body, der ja die Service Methode identifiziert, abgearbeitet werden und bei einem eventuellen Nichtverstehen sofort ein SOAP Fault ausgelöst werden. Dies ist aber hier offensichtlich nicht der Fall. Man muss aber in seine Überlegungen miteinbeziehen, dass alle Vorgänger des Apache Axis Projekts dieses Attribut überhaupt nicht unterstützt haben. Erst im Axis Projekt wurde eine rudimentäre Unterstützung für das *mustUnderstand*-Attribut eingebaut. Zu erwähnen wäre, dass die oben erwähnte Methode `setProcessed()` nicht dokumentiert ist und erst nach genauerem Studium des Quellcodes der entsprechenden Klasse sich die Funktionalität dieser Methode erschlossen hat. Dieser Umstand stellt schon einen gewaltigen Minuspunkt dar. Dokumentation ist für die Benutzung dieses Teils der SOAP Nachricht noch unbedingt erforderlich.
Um dieser potentiellen Fehlerquelle zu entgehen, wäre es von Serviceentwicklersicht her besser, wenn man gleich das Handlerkonzept für die Verarbeitung von SOAP Headern in sein Service miteinbezieht. Die JAX-RPC Spezifikation sieht dieses Konzept vor. Deshalb wird es auch in Axis umgesetzt. Eine konkrete Beschreibung dieses Verarbeitungsmodells findet sich etwas später in diesem Abschnitt, wenn es um die Verarbeitung von SOAP Headern in der JAX-RPC Referenzimplementierung geht.
Auch im .NET Framework stellen SOAP Header einen essentiellen Teil der SOAP Nachricht dar. Der Zugang und das Erstellen von SOAP Headern sind auch hier ein wichtiger Teil bei der Implementierung einer Server- oder Client-Applikation.
In diesem Absatz soll die serverseitige SOAP Header-Programmierung des ASP.NET Frameworks näher beleuchtet werden. Die Schritte zur Erstellung eines SOAP Headers in einem SOAP Service, das sich auf ASP.NET stützt, sind recht einfach. Das Hauptelement bei dieser SOAP Implementierung ist die Klasse `System.Web.Services.Protocols.SoapHeader`, die die Basisfunktionalität für SOAP Header beinhaltet und von der sich jeder SOAP Header eines Services ableiten muss. Konkret sehen die Schritte für die Erstellung eines SOAP Headers in einer Service Klasse so aus ([7]):

1. Erstellen einer Klasse, die sich von der oben erwähnten System-Klasse ableitet. Es ist gewährleistet, dass die erstellte Klasse dann entsprechend den Vorgaben serialisiert wird.

2. Hinzufügen einer Variable des im ersten Punkt erstellten Typs. Der Zugriff auf die SOAP Header Variable auf Klassenebene ist gesichert. Die Variable dient zum Setzen beziehungsweise Auslesen des generierten SOAP Headers.

3. Setzen des `[SoapHeader]` Attributs für die Service Methoden, die den SOAP Header benutzen sollen. Mittels des Attributs wird bekannt gegeben, auf welche SOAP Header Variable sich die Methode eigentlich bezieht.

4. Benutzen der Objektvariablen zum Lesen und Schreiben der SOAP Header Information.

Interessant bei dieser Implementierung der SOAP Spezifikation ist die Lösung des Problems mit dem *mustUnderstand*-Attributs eines SOAP Headers. Ist nämlich dieses Attribut für einen SOAP Header in der Nachricht gesetzt und existiert eine Variable mit entsprechender Klasse für diesen SOAP Header in der Serviceklasse, so wird vom Framework automatisch angenommen, dass das Service diesen Header verarbeiten kann. Wird eine Nachricht mit einem SOAP Header empfangen, der das `mustUnderstand` Attribut ebenfalls gesetzt hat, aber keine eigene SOAP Header Klasse für diesen existiert, so wird dieser Header Block in der `SoapUnknownHeader` Klasse gespeichert. Mit dieser Klasse hat man dann die Möglichkeit, auf die XML-kodierten Informationen eines unbekannten SOAP Headers zuzugreifen. Hat man diesen SOAP Header verstanden und bearbeitet, so muss man die `DidUnderstand` Eigenschaft der Klasse auf `true` setzen, sonst wird vom Framework her ein SOAP Fault ausgelöst. Problematisch bei diesem Ansatz ist, dass die Servicemethode bis zum Ende ausgeführt wird und dann erst vom Framework überprüft wird, ob alle SOAP Header der eingegangenen Nachricht verstanden wurden. Dieses Verhalten ist zwar durch die Ungenauigkeit der SOAP 1.1 Spezifikation gedeckt, aber in der Spezifikation zu SOAP 1.2 nicht mehr erlaubt. Man darf also gespannt sein, wie Microsoft diese Unstimmigkeit in der kommenden Version des ASP.NET Pakets ausräumen wird.
Die Unterstützung für SOAP Header bei der clientseitigen Programmierung von Applikationen läuft ähnlich ab. Problematisch ist bei diesem Punkt nur, dass SOAP Header in der WSDL-Beschreibung eines Services explizit angegeben werden müssen, damit sie bei der automatischen Proxy-Generierung berücksichtigt werden. Ist dies nicht der Fall, ist Handarbeit von Nöten, um

die noch fehlenden SOAP Header in der Quelldatei des Serviceproxys einzutragen. Die Erstellung von SOAP Headern auf der Clientseite folgt den Schritten auf der Serverseite.
Will man auch bei .NET Remoting Diensten das SOAP Header Feature benutzen, ist der Aufwand ungleich höher. Da bei .NET Remoting nur eingeschränkt vorgesehen ist, dass der Entwickler Einfluss auf die Serialisierung eines Objekts nimmt, kann man SOAP Header nur in den Stream eines serialisierten Objekts schreiben. Der Aufwand, um so einen SOAP Header wieder aus dem Stream zu extrahieren, ist ebenfalls beträchtlich. Zugriff auf den Stream beziehungsweise auf die SOAP Header ist über den sogenannten `CallContext` möglich. Eine einfache und direkte Manipulation von SOAP Headern ist in .NET Remoting nicht vorgesehen.
Ein Problem bezüglich SOAP Headern bei .NET Remoting gibt es da noch, das auch in das Thema der Interoperabilität und der Umsetzung des SOAP Standards fallen würde. Erzeugt man nämlich mittels Remoting Architektur einen SOAP Header in einer Nachricht, so wird dieser Header standardmäßig mit einem gesetzten *mustUnderstand* Attribut initialisiert. Es ergibt sich aber ein Problem, denn der Standardwert für ein *mustUnderstand* Attribut ist nämlich, dass es nicht gesetzt ist. Somit erfolgt hier ein grober Verstoß gegen die SOAP/1.1 Spezifikation. Dieses Verhalten lässt sich vielleicht dadurch erklären, dass der Entwickler eigentlich keinen direkten Einfluss auf SOAP Header bei .NET Remoting haben sollte ([63]).
Bei der JAX-RPC Implementierung ist es in erster Linie Handlern vorbehalten, auf die SOAP Header Elemente einer SOAP Nachricht zuzugreifen. Diese haben mittels dem SOAP Messagekontext die Möglichkeit, alle SOAP Header einer Nachricht einzusehen. Diese Header einer Nachricht können in einer Schleife abgearbeitet werden. Dazu gibt es noch eine Methode, die nur die SOAP Header liefert, die der Rolle des aktuellen SOAP Knotens entsprechen. Dies erlaubt ein gezieltes Auslesen aller SOAP Header. Es ist aber notwendig SOAP Header mit gesetztem *mustUnderstand* Attribut schon frühzeitig zu erkennen und dann eventuell mit einem SOAP Fault zu quittieren. Die SOAP Header, die von der Applikation verarbeitet werden können, müssen bereits vorab der Laufzeitumgebung bekanntgeben werden. Nur so ist ein problemloses Verarbeiten von SOAP Headern möglich. Da man aber auch auf der Clientseite eine zusätzliche Handlerklasse braucht, um auf die SOAP Header einer Nachricht zugreifen zu können, ist dieser Zugriff in einer eigenen Klasse gekapselt. Nähere Informationen zur Umsetzung von Handlern für SOAP Header sowohl auf Client- als auch auf Serverseite finden sich in [61].
Es ist aber auch bei der JAX-RPC Referenzimplementierung möglich, direkt in der Service-Klasse auf den Inhalt eines SOAP Headers zuzugreifen. Dafür ist es aber notwendig, dass der SOAP Header in der WSDL Beschreibung der

Servicemethoden definiert wurde. Der SOAP Header wird dann zum Service Endpoint Interface als Parameter der spezifizierten Methode hinzugefügt. Deshalb ist auch die Angabe des zu verarbeitenden SOAP Headers bei der WSDL Beschreibung der Servicemethode unbedingt notwendig. SOAP Header, die nicht im Vorhinein klar definiert wurden, können von einer Servicemethode auf diesem Weg nicht verarbeitet werden.
Die beiden genannten Möglichkeiten sind sowohl auf der Client- als auch auf der Serverseite vorhanden. Das Konzept von Nachrichtenhandlern ist bei JAX-RPC auch bei Clientanwendungen realisiert. Es ist ebenfalls möglich, mittels Handlern die SOAP Header einer Nachricht zu setzen beziehungsweise zu verarbeiten.
Es gibt aber noch eine weitere Möglichkeit, die es einem Service erlaubt, die empfangenen SOAP Header einzusehen. Mittels des `ServiceLifecycle` Interfaces wird der Serviceimplementierung der aktuelle Nachrichtenkontext übergeben. Aus diesem lassen sich dann auch die SOAP Header leicht extrahieren. Dies stellt eine Vermischung der beiden oben erwähnten Methoden dar und lässt sich nur auf der Serverseite durchführen. Eine genauere Beschreibung der aufgezählten Möglichkeiten mittels JAX-RPC SOAP Header zu verarbeiten findet sich in [62].
Die Verarbeitung von SOAP Header ist in der JAX-RPC Referenzimplementierung fast so einfach möglich, wie in den beiden anderen SOAP Kommunikationsplattformen. Es wäre aber hier ein Konzept gegeben, um SOAP Header mit gesetztem *mustUnderstand* Attribut schon frühzeitig zu erkennen und dann entsprechend reagieren zu können. Dies wäre dann auch im Sinne der Spezifikation von SOAP 1.2.

4.3.2 Umsetzung von SOAP Attachments

SOAP Attachments stellen eine sehr nützliche Erweiterung für SOAP Kommunikationsplattformen dar. In diesem Abschnitt soll untersucht werden, in wie weit die vorgestellten SOAP Kommunikationsplattformen dieses SOAP Feature unterstützen, beziehungsweise wie die Implementierungsdetails für die zu nutzenden Funktionalitäten ausschauen.
Wie in Abschnitt 2.11 bereits beschrieben, stellt das SOAP Attachment Feature eine sinnvolle Erweiterung einer SOAP Kommunikationsplattform dar.
Mit der Basisinstallation des .NET Frameworks ist es in ASP.NET nicht möglich, SOAP Nachrichten mit Attachments zu verschicken, da dieses Feature nicht standardmäßig dabei ist. Es gibt zwar die Möglichkeit, dass man Binärdaten in die SOAP Nachricht verpackt und diese Base64 kodiert, aber dieser Weg ist wenig sinnvoll. Es kommen dadurch alle in Abschnitt 2.11 angeführten Probleme zum Tragen. Es gibt aber auch in diesem Punkt Abhilfe.

Wird das Web Service Enhancement Paket (WSE) installiert, so steht ein Mechanismus zur Verfügung, der DIME-kodierte Nachrichten erzeugen kann, die auch SOAP Nachrichten enthalten können. Mittels diesem Zusatz ist es im ASP.NET Framework möglich, das SOAP Attachment Feature zu nutzen. Microsoft verwendet für diesen Zweck einen eigens ausgearbeiteten Standard. Dieser wird in [52] beschrieben. Zur Zeit der Erstellung dieser Arbeit befand sich Microsoft in puncto SOAP Attachments gerade in einem Umbruch. Die Unterstützung für DIME-kodierte Nachrichten hat sich als nicht mehr zielführend herausgestellt und wird in nächster Zeit nicht mehr weiterverfolgt werden. Unter Mitarbeit von Microsoft kristallisiert sich momentan eine neue Spezifikation heraus, die konformer mit der SOAP Spezifikation sein soll, als es im Moment DIME- aber auch MIME-kodierte Nachrichten sind. Dieser neue Standard wird SOAP Message Transmission Optimization Mechanism (MTOM) genannt und ist in [41] näher beschrieben. In dieser Arbeit wurde bereits in Abschnitt 2.11 auf diese Spezifikation eingegangen.
Mittels der .NET Remoting Architektur ist es nicht vorgesehen, SOAP Attachments zu verschicken. Dieses Feature wird von dieser Implementierung nicht unterstützt, da es für diesen Zweck einen BinaryFormatter gibt.
Die JAX-RPC Referenzimplementierung von Sun baut bei dem SOAP Attachment Feature auf den von ihnen entworfenen SAAJ (SOAP Attachment API for Java) auf. Dieses API setzt auf der MIME-Technologie und stützt sich auf die „SOAP with Attachment“ Spezifikation in [10]. Mittels dieser Implementierung ist es möglich, SOAP Attachments zu generieren. Die Technologie der MIME-kodierten Nachrichten fand auch Einzug in das Attachment Profil der WS-I. Dadurch ist die JAX-RPC Referenzimplementierung konform zu diesem Profil. Dies erstaunt doch, da Microsoft schon angekündigt hat, dieses SOAP Attachment Format in keiner seiner Implementierungen mehr einzubringen und schon auf die kommende SOAP MTOM Spezifikation setzt ([57]). Durch diesen Umstand wären dann die SOAP Attachment Implementierungen des .NET Frameworks nicht mehr Profil-konform. Dies ist ein Indiz dafür, dass Sun Microsoft in der WS-I bereits den Rang abgelaufen hat.
Vergleicht man die beiden genannten Umsetzungen, nämlich MIME- und DIME-kodierte Nachrichten, so bemerkt man gleich, dass diese beiden auf unterschiedlichen Technologien fußen, die zueinander nicht kompatibel sind. Dies bedeutet, dass sich zwischen dem .NET Framework und der JAX-RPC Referenzimplementierung keine SOAP Attachments austauschen lassen, da die Kodierungen der Nachrichten unterschiedlich sind. Es wäre auch hier wieder ein Interoperabilitätsproblem zu lösen.
Anders ist die Sachlage beim Apache Axis Projekt. Da Axis die Schnittstellen der Java XML RPC Spezifikation implementiert hat, musste auch das SAAJ API umgesetzt werden. Es ist grundsätzlich möglich, MIME-kodierte

Nachrichten, die SOAP Attachments enthalten, zu erstellen beziehungsweise zu verarbeiten. Will man SOAP Attachments mittels dem Axis Framework verschicken, so stellt die MIME-kodierte Nachricht auch die Standardeinstellung dieser SOAP Implementierung dar. Es ist aber auch möglich, DIME-kodierte Nachrichten zu verarbeiten. Dazu ist es nötig, die Einstellung für die Verarbeitung zu ändern. Grundsätzlich steht einer Kommunikation zwischen ASP.NET und der Apache Axis SOAP Implementierung mittels DIME-kodierten Nachrichten nichts im Weg. Axis ist somit die einzige SOAP Kommunikationsplattform in diesem Test, die sowohl MIME- als auch DIME-kodierte Nachrichten verarbeiten kann. Durch diese zusätzliche Funktionalität hebt sich diese SOAP Implementierung stark von den beiden anderen ab.
Wie der oben gezeigte Vergleich verdeutlicht, fehlt es nicht an Umsetzungen des SOAP Attachment Features. Leider gibt es in diesem Bereich aber zwei konkurrierende Spezifikationen mit ihren jeweiligen Unterstützern. Es ist aber nicht möglich, SOAP Attachments zwischen diesen konkurrierenden Technologien auszutauschen. Im wesentlichen besteht der Unterschied zwischen MIME und DIME darin, dass die Trennung der einzelnen Nachrichtenteile anders gehandhabt wird. Bei MIME wird eine spezielle Zeichenfolge als Abgrenzung verwendet, während bei DIME pro Nachrichtenteil ein Headereintrag erstellt wird, der die Länge jedes Teils angibt. Beide Nachrichtenformen sind eigentlich keine wirklichen SOAP Nachrichten sondern beinhalten nur diese Art von Nachrichten. Auf einen ähnlichen Aufbau vertraut auch die SOAP MTOM Spezifikation. Hier wird die SOAP Nachricht ebenso in einer kodierten Nachricht mit Attachments versendet. Jedoch wird diese Nachricht nur für den Transport kodiert und direkt nach dem Transport wieder in eine einzige SOAP Nachricht mit Binärteilen umgewandelt. Dadurch lassen sich Binärdaten ohne Probleme versenden, aber der Aufbau der kodierten Nachricht wird transparent über das Transportprotokoll abgewickelt. Für das Service bleibt es immer eine SOAP Nachricht mit binärer Information.
Da es sich bei Sun und Microsoft um zwei große Rivalen handelt, bleibt abzuwarten, ob und wann der MTOM Standard sich durchsetzen wird. Auf jeden Fall gibt auch in diesem Bereich im Moment ein erhebliches Interoperabilitätsproblem. Einzig Axis hat es verstanden, MIME- und DIME-kodierte Nachrichten umzusetzen. Die Unterstützung von SOAP Attachments ist also sehr stark von der verwendeten SOAP Kommunikationsplattform abhängig, was einen plattformunabhängigen Austausch dieser Binärdaten sehr erschwert.
Es bleibt abzuwarten, wie die neue Spezifikation von den Entwicklern der SOAP Kommunikationsplattformen angenommen wird.

4.3.3 Transportprotokolle

Grundsätzlich unterstützen alle vorgestellten SOAP Kommunikationsplattformen HTTP als Transportprotokoll. Dies ist auch der Standard, den es als Mindestanforderung zu implementieren gilt.
Da SOAP aber eigentlich Transportprotokoll unabhängig definiert wurde, sollte man auch andere Protokolle zum Transport verwenden können. Dies ist aber in der Realität nicht wirklich gegeben. Außer für HTTP wurde nur für SMTP noch ein SOAP Binding in einer W3C Note definiert. Andere SOAP Bindings sind zwar vorstellbar, aber nicht wirklich durchführbar, da dazu die nötige Standardisierung fehlt. Es gibt zwar einzelne alternative Umsetzungen von SOAP Bindings, aber diese funktionieren dann nicht mehr über SOAP Kommunikationsplattformen hinweg, sondern sind auf eine bestimmte Implementierung beschränkt.
Wir wollen uns jetzt die unterstützten Protokolle des Apache Axis Projekts noch einmal anschauen. Das Standardprotokoll für das Verschicken und Empfangen von SOAP ist HTTP. Innerhalb des Apache Axis Projekts hat sich auch die Möglichkeit gebildet, SMTP zum Transport von Nachrichten zu verwenden. Es gibt eine Beispielimplementierung, die es erlaubt, sowohl E-Mails, die SOAP Nachrichten enthalten, zu empfangen als auch diese zu versenden. Der mitgelieferte Beispielserver ist aber für Produktionszwecke nicht geeignet und sollte daher eher für Demonstrationen und Performance Profiling benutzt werden.
Ein weiteres interessantes Protokoll, das von der Axis Implementierung unterstützt wird, ist JMS (kurz für Java Messaging Service). Mittels diesem Protokoll ist es möglich, verlässliche Transportkanäle für SOAP Nachrichten zu benutzen. Axis bietet einen Zugang zu einer Alternative für HTTP, die restlos in das Axis Framework eingebettet ist. Man greift auf den verlässlichen Zustellmechanismus der JMS Messaging Semantik zurück. Problematisch ist, dass man an beiden Enden der Kommunikation über JMS Axis Komponenten einsetzen muss und somit auf SOAP Kommunikationsplattform-Unabhängigkeit verzichten muss. Dieses Protokoll stellt aber eine interessante Alternative zu HTTP dar.
Im Axis Projekt gibt es auch Bestrebungen, nur TCP als Transportprotokoll für SOAP Nachrichten zu verwenden. Diese Umsetzung ist rein experimentell und noch im Entwicklungsstadium. An eine breitere Anwendung dieser Alternative ist hier nicht zu denken ([3]).
Bei der Sun Referenzimplementierung zur JAX-RPC Spezifikation hat man sich rein auf eine Umsetzung des HTTP Bindings für SOAP fokussiert. Alternative Transportprotokolle sucht man in dieser SOAP Kommunikationsplattform vergebens.

Beim ASP.NET Framework ist auch die HTTP Umsetzung des SOAP Bindings der Standard. Allerdings gibt es mittels des Web Service Enhancement Pakets die Möglichkeit, TCP als Transportprotokoll für Services zu nutzen. Dieses System ist im Gegensatz zur Implementierung von Axis schon ausgereifter und kann auch in Produktionsystemen genutzt werden. Von der Handhabung im Code unterscheiden sich TCP-basierte Services nur marginal. Wie bereits in Abschnitt 3.3.3 beschrieben, verfügt auch die .NET Remoting Architektur über die Möglichkeit, SOAP Nachrichten mittels TCP zu verschicken.
Es hat sich gezeigt, dass die aktuellen SOAP Kommunikationsplattformen einige alternative Transportprotokolle unterstützen, aber im Gegenzug diese Protokolle zur Bindung an eine bestimmte SOAP Implementierung führen. Die unterstützten Alternativen sind von Implementierung zu Implementierung doch recht unterschiedlich, was eine Plattformunabhängigkeit sehr erschwert. Eine echte Alternative zu HTTP hat sich hier nicht herauskristallisiert, womit man gespannt sein darf, was in nächster Zukunft als eine allgemein akzeptierte Alternative des Transports von SOAP Nachrichten präsentiert werden wird.

4.3.4 Sessions

Der Begriff „Session“ bedarf etwas Erläuterung. Im Allgemeinen wird mit einer Session eine Reihe von Nachrichten zwischen zwei Parteien gemeint, die eine bestimmte Korrelation haben. Man kann es durchaus mit einem Gespräch zwischen zwei Kommunikationspartnern vergleichen.
Sessions sind für den Entwurf von Services sehr wichtig. Auch bei SOAP Nachrichten ist es oft notwendig, dass man eine Beziehung zwischen verschiedenen Nachrichten herstellen kann. Aufbauend auf diese Technologie lassen sich dann auch Mittel zur Unterstützung der Verlässlichkeit eines Services implementieren. Verlässlichkeit ist bei Services insofern ein Thema, als die aktuell vorliegenden SOAP Kommunikationsplattformen standardmäßig SOAP Nachrichten über HTTP verschicken. Dieses Protokoll an sich hat aber keine Mittel, um Verlässlichkeit auf der Ebene des Transportprotokolls zu definieren. Es gibt zwar die Initiative HTTPR, siehe dazu Abschnitt 4.2.6, aber es bleibt die Frage, ob Verlässlichkeit nicht auch eine Erweiterung für eine Applikation sein sollte, also die Daten dafür direkt in der SOAP Nachricht transportiert werden sollten ([47, 38]). Auf jeden Fall wären mit Sessions schon eine Grundlage für diese Technologie geliefert.
Problematisch ist, dass es im Moment noch keinen regulären Standard für Sessions bezüglich SOAP Kommunikation gibt. Die meisten SOAP Implementierungen haben zwar eine Art dieses Mechanismuses definiert, aber es

gibt keinen allumfassenden Standard, der die Entwicklung in eine Richtung vorantreiben würde. Ideen in diese Richtung gibt es schon ([42]).
Apache Axis unterstützt in der Version 1.1 zwei Arten von Sessions. Zum einen wären dies sogenannte Cookie-Sessions. Die Information über eine Session wird mittels einem HTTP-Header mit der SOAP Nachricht versandt. Um diese Art von Sessions zu unterstützen, müsste eine Client-Anwendung fähig sein, mit Cookies umzugehen. Der Server-Anwendung ist es möglich, Daten in Session-Variablen zu speichern und diese zu einem späteren Zeitpunkt wieder auszulesen. Dieser Ansatz findet häufig in Web-Applikationen Anwendung. Eine solche Session wird aber zu jeder Zeit nur vom Server verwaltet. Der Client hat keinen direkten Einfluss auf die Session und kann diese somit auch nicht wirklich initiieren beziehungsweise beenden. Diese Aktionen bleiben rein dem Server vorbehalten. Es bleibt noch zu erwähnen, dass die Anwendung dieser Session-Technologie nur auf HTTP als Transportprotokoll für SOAP Nachrichten beschränkt ist.
Axis bietet aber auch noch eine zweite Art. Die Informationen zur Session werden im SOAP Header der Nachricht transportiert. Dieser Mechanismus wurde sehr flexibel gestaltet und erlaubt es deshalb, leicht angepasst zu werden. Da es aber, wie bereits vorher erwähnt, keine Standardisierung auf diesem Gebiet gibt, ist die Session-basierte Kommunikation mittels SOAP Headern nur zwischen Axis Kommunikationsplattformen möglich.
Bei der Referenzimplementierung von JAX-RPC ist es ebenfalls möglich, Sessions zu nutzen. Es werden bei dieser Implementierung Funktionen der Klasse `javax.servlet.http.HttpSession` ausgenützt. Man verlässt sich also auf die Session-Funktionalität, die vom Servlet Container zur Verfügung gestellt wird. Es ist bei der Erstellung eines `Call`- beziehungsweise `Stub`-Objekts möglich, eine Eigenschaft zu setzen, die bewirkt, das der Sessionkontext aufrechterhalten bleibt. Eine erweiterte Form von Sessions über SOAP Header, wie es sie beim Apache Axis Projekt gibt, sucht man bei dieser SOAP Kommunikationsplattform aber wieder vergebens.
Auch in ASP.NET gibt es die Möglichkeit, bei Diensten Sessionfunktionalität zu nutzen. Die Nutzung ist auch hier auf HTTP Sessions beschränkt. Das Session-Objekt ist nur auf der Serverseite verfügbar und kann in einer Clientapplikation nicht manipuliert werden. Darüber hinaus gibt es in ASP.NET keine wie auch immer geartete Möglichkeit Sessions zu nutzen. Eine Erweiterung des Session-Mechanismuses müsste selber entwickelt werden, da das .NET Framework in der aktuellen Version diese Möglichkeit nicht in Betracht zieht. Eine gelungene Anleitung, wie eine SOAP Header-basierte Architektur zu entwerfen ist, findet man in [7]. Hier wird ein Weg gezeigt, wie eine möglichst erweiterbare Struktur für Sessions aussehen könnte.
Aus diesem Vergleich ergibt sich, dass es zum Standard der vorgestellten

SOAP Kommunikationsplattformen gehört, HTTP Sessions in der Implementierung zu unterstützen. Da diese aber nur auf die Serverseite beschränkt sind und eigentlich vom Transportprotokoll der SOAP Nachricht zur Verfügung gestellt werden, kann dies nur eine Übergangslösung sein. Es müssen Mittel und Wege gefunden werden, wie die Sessioninformation in die SOAP Header einer Nachricht verpackt werden. Nur mit so einer flexiblen Lösung ist es möglich, auch wirklich Transportprotokoll unabhängige Session-Funktionen der Applikation zu erstellen.
Allein Apache Axis bietet bereits einen Weg, Sessioninformation über SOAP Header zu führen. Problematisch ist hier, dass es sich um keinen umgesetzten Standard handelt, sondern diese Funktionalität auf Eigeninitiative des Projekts zurückgeht. Viel wichtiger in diesem Bereich wäre ein genereller Standard, der angibt, wie ein Session Header in einer SOAP Nachricht auszusehen hat.

4.4 Sicherheitsüberlegungen

Die Sicherheit von Services, die über SOAP Nachrichten kommunizieren, ist eine sehr heikle Angelegenheit. Sicherheit an sich ist im Vergleich zur Implementierung eines Services sehr schwierig zu bewerkstelligen ([7]).
In den ersten Versionen von SOAP Kommunikationsplattformen spielte Sicherheit nur eine untergeordnete Rolle. Ernsthafte Überlegungen in diese Richtung wurden erst in letzter Zeit angestellt ([36, 56, 1]). Dieses Phänomen ist aber bei der gesamten Entwicklung des Web Service Stacks zu erkennen. Sicherheit wird zwar als eines der grundlegenden Elemente eines entfernten Services angesehen, aber die wirkliche Umsetzung von Konzepten zur Implementierung beziehungsweise Erhöhung der Sicherheit wird erst nach der Umsetzung der meisten anderen Features angegangen. Es sollte besser Hand in Hand gehen. Aufgepflanzte Sicherheitsmechanismen sind meist problemanfällig und schwer integrierbar ([2]).
In diesem Abschnitt soll untersucht werden, welche Konzepte in puncto Sicherheit in den einzelnen SOAP Implementierungen umgesetzt wurden, um auch vertrauliche Daten mit einem entfernten Dienst verarbeiten zu können. Eine grundlegende Sicherheitsmaßnahme bei allen SOAP Implementierungen ist, dass man das Transportprotokoll dazu verwendet, um verschlüsselte Kommunikation zu bewerkstelligen. Dieser Ansatz wird nach wie vor von den meisten SOAP Implementierungen verwendet. Diese Funktionalität hat aber nichts mit SOAP oder dem Service an sich zu tun, sondern wird als Dienst von der Transportschicht der SOAP Implementierung zur Verfügung gestellt. Ein Beispiel dafür ist der SSL (Secure Socket Layer) über HTTP. Mittels dieser Technik wird der Datenverkehr verschlüsselt. Dieser Mechanismus ist aber

sowohl für die Client- als auch für die Server-Applikation transparent, da diese Funktionalität auf Serverseite meistens das Hosting der jeweiligen SOAP Kommunikationsplattform übernimmt. Dies wäre bei Apache Axis der Servlet Container, wie zum Beispiel Tomcat, der verschlüsselte Kommunikation zulässt. Gleiches gilt in diesem Zusammenhang aber auch für die JAX-RPC Referenzimplementierung, die ebenfalls mittels Apache Tomcat genutzt werden kann. Beim ASP.NET Framework wird diese Art der Verschlüsselung durch den IIS zur Verfügung gestellt. Auf der Clientseite ist meist die Klasse, die für die HTTP-Verbindung zuständig ist, auch fähig, mittels HTTPS zu kommunizieren. Wie bereits erwähnt, ist diese Art von Sicherheit aber nur auf das Transportprotokoll beschränkt.
Mit diesem Ansatz ist es nicht möglich nur einen Teil der SOAP Nachricht zu verschlüsseln, während andere Teile, die nicht vertraulich sind, ausgespart werden können. Bezüglich Sicherheit von SOAP Nachrichten gibt es noch mehr. Diese Arbeit soll sich nur auf Sicherheitsmaßnahmen beschränken, die sich auf die Nachrichtenebene beziehen. Diese Mechanismen schaffen es auch, nur Teile einer Nachricht zu verschlüsseln, den Absender einer Nachricht zu authentifizieren oder die Authentizität einer Nachricht mittels digitaler Signatur zu bestätigen.
Grundsätzlich gilt es einmal zu klären, was Sicherheit eigentlich als Basisfunktionalität zur Verfügung stellen muss. Im Allgemeinen steht der Begriff meist für drei eigentlich sehr unterschiedliche Bereiche, die sich unter dem Aspekt der Sicherheit vereinigen lassen. Diese drei Teilbereiche der Sicherheit wären ([7]):

- Authentifizierung: Dies bedeutet, dass nur autorisierte Benutzer Zugriff auf bestimmte Dienste und Ressourcen haben. Dieser Bereich wird oftmals in zwei weitere Bereiche unterteilt. Das wäre zum einen die Technologien zum Versenden der Identität eines Benutzers, üblicherweise Authentifizierung genannt. Zum anderen wäre dies die Autorisierung. Man meint damit im Allgemeinen Technologien, die es erlauben, Benutzern bestimmte Rechte in Bezug auf das Service zuzuordnen.

- Vertraulichkeit: Mit Vertraulichkeit ist gemeint, dass Daten, die man einem Service sendet, sicher übertragen werden. Üblicherweise verwendet man für diesen Zweck Verschlüsselung. Dafür werden zwei Hauptformen der Verschlüsselung zur Verfügung gestellt. Erstens wäre hier die symmetrische Verschlüsselung zu nennen. Ein Schlüssel wird zur Ver- und Entschlüsselung benötigt. Eine zweite Form der Verschlüsselung stellt Public Key Encryption dar, wobei es hier ein Schlüsselpaar gibt.

- Integrität: In diesem Punkt geht es darum, ob eine Nachricht, während

sie an den Empfänger übertragen wurde, verändert worden ist. Es soll sichergestellt werden, dass kein Dritter Einfluss auf den Inhalt einer Nachricht nimmt.

Im nächsten Abschnitt sollen Standards präsentiert werden, die sich auf Sicherheit von Services im Bereich von SOAP Nachrichten beziehen.

4.4.1 Sicherheitsstandards

Im Moment ist man beim W3C beschäftigt, Verschlüsselung für XML-Daten zu definieren. XML Encryption wird in [44] näher spezifiziert.
XML Encryption erlaubt es, drei verschiedene Bereich eines XML Dokuments zu verschlüsseln. Diese drei Bereiche wären das gesamte Dokument, nur ein Knoten des Dokuments oder einfach nur der Wert eines XML Elements. Durch diese Technik lassen sich sehr feine Abstufungen bei der Verschlüsselung eines Dokuments treffen.
Zum Zwecke der Integrität wird beim W3C auch an der Möglichkeit der Signatur eines XML Dokuments gearbeitet. Eine Recommendation ist schon auf dem Weg. Die Spezifikation zu XML Signature findet sich in [9]. Zu diesem Zweck wird Public Key Kryptographie verwendet. Es wird hierzu ein Hash-Wert für ein Dokument oder eine Nachricht ermittelt, der mittels einem Private Key verschlüsselt wird. Diese Art der Verschlüsselung erfüllt gleich zwei Aufgaben, denn es werden hiermit der Sender einer Nachricht authentifiziert, in dem der Public Key des Senders benutzt wird um den Hash-Wert zu entschlüsseln. Zum anderen wird mittels des entschlüsselten Hash-Werts verglichen, ob sich die Nachricht nach dem Absenden verändert hat. Dies ist nämlich der Fall, wenn der entschlüsselte Hash-Wert und der neu berechnete nicht übereinstimmen ([55]).
Diese beiden Technologien werden in einer anderen Spezifikation, die im folgenden Abschnitt näher erläutert werden soll, eingesetzt.

4.4.1.1 WS-Security

Im Frühjahr 2002 haben Microsoft, IBM und Verisign gemeinsam eine neue Spezifikation herausgegeben, die WS-Security getauft wurde. Sinn dieser Spezifikation war es, die Sicherheitsbereiche bei Web Services zu identifizieren und zu vereinheitlichen.
Für die Umsetzung der Spezifikation wurde die Erweiterbarkeit von SOAP benutzt. Die Grundlage für diese Technologie, die hinter der Spezifikation steckt, ist nämlich nur ein einziger SOAP Header-Block, der einer Nachricht hinzugefügt werden muss. Dieser Header-Block wird *Security* genannt und enthält alle notwendigen sicherheitsrelevanten Daten, die von der Applikation

zu verarbeiten sind. Der Header darf zwar mehrere Male in einer Nachricht vorkommen, muss aber dann durch unterschiedliche SOAP Actor Attribute unterscheidbar sein. Natürlich darf ein solcher Header auch einmal ohne Actor Attribut vorkommen ([24]).
Mit dieser Vorkehrung soll sichergestellt werden, dass man für mehrere Intermediaries solche Header definieren kann, wenn eine Nachricht über mehrere Stationen laufen muss. Es ist sogar möglich, dass solche Intermediaries ebenfalls *Security* Header zu einer Nachricht hinzufügen.
Die Spezifikation definiert drei Typen von Daten, die mittels einem solchen *Security* Header-Block versandt werden können. Dies wären sogenannte Security Tokens, Signaturen und verschlüsselte Daten.
Für den Bereich der Signaturen wird empfohlen, die XML Signaturen Spezifikation des W3C zu verwenden, obwohl die tatsächliche Umsetzung in diesem Bereich der Spezifikation eher offen gelassen wurde ([24]). Gleiches gilt im Übrigen auch für die Verschlüsselung. Hier wird darauf hingewiesen, dass man den XML Encryption Standard des W3C verwenden sollte. Die Security Token können alles Mögliche sein. Üblicherweise werden aber Benutzername und Passwort verschickt, wobei es beim Passwort-Element möglich ist, den Inhalt Klartext oder gehasht zu verschicken. Es ist ebenfalls möglich, dass man binäre Security Token wie etwa Zertifikate versendet.
Da Microsoft einer der Initiatoren der WS-Security Spezifikation war, ist anzunehmen, dass das aktuelle .NET Framework diesen Standard unterstützt. Dies ist aber nicht der Fall. Um die in der Spezifikation vorkommenden Security Features nützen zu können, ist es nämlich notwendig, dass man neben dem .NET Framework auch noch das Web Service Enhancement Paket installiert. Erst mit diesem Zusatz ist es möglich, die oben erwähnten sicherheitsrelevanten Zusätze zu benutzen, obwohl die Verschlüsselungsmethoden schon standardmäßig im Framework enthalten sind. Hat man aber das WSE Paket installiert, dann ist der Zugriff und das Erstellen dieses *Security* Headers für den Entwickler sehr einfach zu bewerkstelligen. Wie bereits früher erwähnt, ist es notwendig, dass das WSE Paket sowohl am Server als auch am Client installiert wurde. Sonst ist die Nutzung dieses Security Features nicht so einfach möglich.
Ähnlich gestaltet sich der Sachverhalt beim Apache Axis Projekt. Auch hier gibt es keine direkte Unterstützung der WS-Security Spezifikation. Eine Umsetzung dieser Spezifikation innerhalb der Apache Gruppe gibt es aber. Diese baut auch auf Apache Axis auf und soll dann JAX-RPC basierte Client- und Serverapplikationen als auch .NET Client- und Serveranwendungen unterstützen ([5]). Das Projekt läuft unter dem Namen WSS4J. Dies ist eine Kurzbezeichnung für WS-Security für Java. Mittels diesem Projekt wird ein API zur Verfügung gestellt, das es erlaubt, SOAP Nachrichten zu signieren

und zu verifizieren in denen WS-Security Informationen transportiert werden. Als die derzeit aktuelle Version des APIs wurde 1.0 angeführt.
Für die Referenzimplementierung der JAX-RPC Spezifikation gibt es im Moment nur einen experimentellen Zugang zu Funktionen der WS-Security Spezifikation. Im Paket ist nämlich ein nicht standardkonformes API enthalten, das die Möglichkeit bietet, SOAP Nachrichten zu signieren beziehungsweise zu verifizieren. Diese Funktionalität basiert im Moment auf der Apache XML-Dsig (XML Digital Signature) Implementierung ([50]). Dieses API wird sich aber in nächster Zeit ändern, da geplant ist, die Implementierung durch eine standardkonforme zu ersetzen. Es werden dem Entwickler Grundfunktionen bezüglich Sicherheit zur Verfügung gestellt.
Bezüglich .NET Remoting und Sicherheit gibt es im Moment nicht viel zu sagen. Eine Möglichkeit um Services zu sichern, die auf diese Technologie aufbauen, ist es, diese mittels einem Internet Information Server von Microsoft zu hosten. Es wäre möglich, SSL für diese Dienste nutzbar zu machen. Eine Alternative dazu wäre, sich einen eigenen sicheren Kanal zu schreiben. Momentan gibt es aber keine fertige Lösung, die in diese Richtung geht. Hier ist also Eigeninitiative gefragt ([63]).
Der Standard bei allen drei untersuchten SOAP Kommunikationsplattformen ist die Verschlüsselung des Datenverkehrs mittels SSL. Eine darüber hinausgehende Art von Sicherheit wird nur mittels Erweiterungen bewerkstelligt. In diesem Zusammenhang tut sich der WS-Security Standard besonders hervor. Dieser bietet zumindest ein Mindestmaß an sicherheitsrelevanter Funktionalität. Schade ist, dass dieser Standard nicht fix Einzug in die SOAP Kommunikationsplattformen gehalten hat. Denn nur als Standardfeature würde es eine breitere Anwendung finden.

5 Schlussbetrachtung

Motivation für diese Arbeit war es, zu untersuchen, wie das Nachrichtenprotokoll SOAP dazu verwendet werden kann, um Nachrichten über Plattformen hinweg auszutauschen. Ziel dieser Arbeit war es, einen Überblick zu schaffen, welche Möglichkeiten SOAP bietet und wie diese Möglichkeiten dann in den SOAP Kommunikationsplattformen umgesetzt beziehungsweise eben nicht umgesetzt wurden.
Interessant waren vor allem die Probleme, die sich durch diese Art des Nachrichtenaustauschs ergeben. Hier ist der große Bereich der Interoperabilität von SOAP Implementierungen gemeint. Dieser Punkt ist im Moment von großer Bedeutung und wird auch in der Zukunft wahrscheinlich immer bedeutender werden, je komplexer die Aufgaben sind, die die Services zu verrichten haben.
Es war von großem Interesse, inwieweit die derzeitigen Implementierungen die derzeit aktuellste SOAP Version unterstützen. Exemplarisch wurden die SOAP Implementierungen von Apache, Sun und Microsoft unter die Lupe genommen. Genauer handelte es sich bei einer der untersuchten Implementierungen um das Apache Axis Projekt für Java in der Version 1.1. Bei Microsofts .NET Framework wurde sowohl auf das Konzept von ASP.NET eingegangen, als auch die Möglichkeit mittels .NET Remoting SOAP Nachrichten zu versenden. Im Apache Axis Projekt wurde die JAX-RPC Spezifikation umgesetzt. Zu dieser Spezifikation gibt es auch eine Referenzimplementierung von Sun in Java. Grund genug, um diese Java SOAP Implementierung näher zu betrachten.
Ziel dieser Arbeit war es, eine Gegenüberstellung der genannten Implementierungen in puncto Interoperabilität und Unterstützung von diversen SOAP Features zu erstellen. Ebenso wurde kurz das Thema der Sicherheit bei diesen SOAP Kommunikationsplattformen angerissen.
Es war notwendig, mittels Literaturrecherche die notwendigen Unterlagen für diese Arbeit zu finden. Dank intensiver Beschäftigung mit diesem Thema ist es gelungen, die Punkte in den untersuchten Plattformen ausfindig zu machen, die heikel beziehungsweise problematisch für den Entwickler sind. Wichtig neben dem Finden der benötigten Literatur war es auch, die identifizierten Probleme mittels praktischer Beispiele näher zu untersuchen beziehungsweise tiefere Einblicke in die Struktur der Problemstellungen zu erlangen. Mittels dieser praktischen Umsetzungen der Service- beziehungsweise Client-Applikationen war erst ein intensives Untersuchen der einzelnen SOAP Implementierungen möglich.

5.1 Resümee zu SOAP 1.2

Ein wichtiger Punkt bei dieser Untersuchung von SOAP Kommunikationsplattformen war, ob und wie die aktuellste SOAP Version 1.2 ihre Umsetzung in den einzelnen untersuchten SOAP Implementierungen gefunden hat. In diesem Punkt bleibt aber festzuhalten, dass es in jeder der drei untersuchten Implementierungen keine bis mangelhafte Unterstützung für SOAP 1.2 gab. Die aktuelle stabile Apache Axis Version 1.1 war die einzige der drei, die wenigstens teilweise SOAP Nachrichten der Version 1.2, aber leider nicht in der aktuellen Version, verschicken beziehungsweise empfangen konnte. Bei den anderen beiden war dies nicht einmal im Ansatz möglich. Dieses Ergebnis enttäuschte, war doch zum momentanen Zeitpunkt die finale Version von SOAP 1.2 bereits über ein Jahr alt. Es gibt aber auch Gutes zu vermelden. Die Nachfolgeversionen aller drei SOAP Implementierungen werden SOAP 1.2-konform sein.

5.2 Die Interoperabilität

Ein Designziel des SOAP Protokolls war es, einen plattformunabhängigen Nachrichtenaustausch zu ermöglichen. SOAP Nachrichten sollten nicht an ein Transportprotokoll gebunden sein, sondern über alle nur erdenklichen Transportkanäle ausgetauscht werden können. Mit diesem Ziel ist der Begriff der Interoperabilität sehr stark verbunden. Denn nur wenn SOAP Kommunikationsplattformen interoperabel sind, ist es möglich, Plattformgrenzen zu überschreiten. Deshalb ist der Interoperabilität und ihren Problemen ein beträchtlicher Teil dieser Arbeit gewidmet worden. Probleme wurden identifiziert, die durch die Ungenauigkeiten der SOAP 1.1 Spezifikation und den damit verbundenen Interpretationen der Implementierungen entstanden sind. Ein Großteil dieser Unstimmigkeiten ließen sich durch die Umsetzung von SOAP Version 1.2 beheben, deren Spezifikation es sich zum Ziel gesetzt hat, bekannte Probleme und Ungereimtheiten der Vorgängerversion zu identifizieren und auszubessern. Da aber SOAP 1.2 noch nicht wirklich umgesetzt wurde, bleibt abzuwarten, wie sich die Probleme in diesem Bereich weiter entwickeln werden.
Ein weiterer Problemherd wurde mit den unterschiedlichen Plattformen an sich identifiziert. Eine Lösung dieses Problems ist auf breiter Basis mittels SOAP kaum bis gar nicht durchführbar. Die Plattformen, welche SOAP verwenden können, sind so unterschiedlich, dass eine Lösung nur auf Applikationsebene Sinn machen würde. Gemeint ist hier vor allem eine Abschätzung von Wertebereichen und Genauigkeit bei Eingabe- beziehungsweise Ausgabeparametern.

Im Moment ist die Interoperabilität von SOAP Kommunikationsplattformen ein heiß diskutiertes Thema. Es haben sich einige Gruppierungen gefunden, die es sich als Ziel gesetzt haben, die Interoperabilität von SOAP Implementierungen voranzutreiben. In dieser Arbeit wurden beispielhaft die SOAP-Builders Gruppe genannt, die mit ihren Interoperabilitätstests viele Fehler beziehungsweise Unzulänglichkeiten beim Austausch von Daten zwischen den Implementierungen aufgedeckt hat. Ebenso wurde die Organisation WS-I angeführt, die aus einer Vereinigung von Unternehmen entstanden ist. Diese hat es sich zum Ziel gesetzt, essentielle Elemente einer Spezifikation zu identifizieren und diese dann in ein Profil zu packen, das es ermöglicht, interoperable Services zu entwerfen. In diesem Zusammenhang wurden Tests entworfen, die überprüfen sollen, ob eine SOAP Implementierung auch der Spezifikation entspricht. Anhand dieser Tests und durch Beratung der WS-I soll es einem Kunden möglich sein, die optimale Implementierung für seinen Zweck zu identifizieren. Ein Hauptkriterium ist natürlich die Interoperabilität.

5.3 Die umgesetzten SOAP Features

Interessant war der Vergleich der untersuchten SOAP Implementierungen bezüglich der jeweiligen Umsetzung von SOAP Features. Diese bieten die Möglichkeit, SOAP Nachrichten entsprechend aufzuwerten, indem zusätzliche Funktionalität zur Verfügung gestellt wird beziehungsweise die Standardfunktionen um sinnvolle Ergänzungen ausgebaut werden. Hier ging es vor allem um Umsetzungen, die die Standard SOAP Implementierungen nicht beziehungsweise nur eingeschränkt bieten.
Eines der untersuchten SOAP Features war das SOAP Attachment Feature, dass eigentlich in jeder untersuchten SOAP Implementierung seine Umsetzung gefunden hat. Bei .NET war es aber wieder einmal nötig, zusätzlich das Web Service Enhancement Paket zu installieren, damit sich diese Möglichkeit erschloss. Dieses SOAP Feature ist nur bedingt über Plattformgrenzen hinweg einsetzbar, da Microsoft und Sun auf zwei verschiedene Standards für die Kodierung dieser Nachrichten setzen. Einzig Apache Axis kann mit beiden Formaten umgehen. Im Moment ist mit SOAP MTOM ein weiterer Standard im Entstehen, der sich auch um dieses Thema kümmert. Microsoft setzt alle seine Hoffnungen in diese Spezifikation ([57]). Man darf also gespannt sein, wie dieser neue Standard die jetzige Situation verändern wird.
SOAP Header stellen ein wertvolles Element in der Struktur einer SOAP Nachricht dar. Durch dieses Nachrichtenelement ist erst die Erweiterbarkeit von SOAP gegeben. Auch für die Implementierung von SOAP Applikationen stellen sie eine Bereicherung dar. Umso wichtiger ist es dann, dass eine SOAP Kommunikationsplattform auch einfachen Zugriff auf diese Elemente erlaubt.

Dies war auch der Aspekt, der für die Untersuchung der gewählten SOAP Kommunikationsplattformen ausschlaggebend war. Es zeigten sich bei den untersuchten SOAP Implementierungen einige Unterschiede. Der Zugriff auf Headerdaten war aber bei allen ohne Probleme möglich. Lediglich die .NET Remoting Architektur sieht keinen einfachen Zugriff auf diese Daten vor. Dies liegt auch am Anwendungsbereich dieser Implementierung. Zu beachten galt die Verarbeitungsreihenfolge von SOAP Headern. Hier liegt bei ASP.NET und teilweise auch Axis das Gewicht beim Entwickler. Es ist dort im Servicecode sicherzustellen, dass alle notwendigen Header verarbeitet wurden, bevor eine Weiterverarbeitung der Servicemethode stattfinden darf.

Kriterium	**Axis**	**Sun RI**	**.NET**
Zugriff auf SOAP Header	+	+	+
SOAP Attachments			
MIME	+	+	-
DIME	+	-	+/-[7]
Sessions			
HTTP Session	+	+	+
SOAP Header Session	+	-	-
Transportprotokolle			
HTTP	+	+	+
SMTP	+	-	-
JMS	+	-	-
TCP/IP	-	-	+/-[8]
Sicherheit			
SSL Verschlüsselung	+	+	+
WS-Security Unterstützung	+/-	+/-	+/-
Interoperabilität			
XML Schema Typunterstützung	+	+/-	+
WS-I Konformität	-	+	+/-[9]

Tabelle 13: Gegenüberstellung der Zusatzfunktionen

Ein weiterer Punkt waren Sessions. Im Moment sind HTTP-Sessions der Standard. Eine erweiterte Möglichkeit mittels SOAP Headern Sessions zu verwalten gibt es nur bei Apache Axis. Es handelt sich dort um keine Umsetzung eines Standards, sondern um eine Eigenentwicklung des Apache Axis Teams. Die Entwicklung eines allumfassenden Standards für diese Thematik

[7] nur mit WSE Paket nutzbar
[8] nur mittels WSE Paket oder Remoting
[9] keine Unterstützung des Basic Attachment Profils

ist unumgänglich. In diesem Bereich werden sich mit Sicherheit in nächster Zeit Neuerungen ergeben.
HTTP ist das Standardtransportprotokoll von SOAP bei den untersuchten Kommunikationsplattformen. Daran wird sich auch in Zukunft nicht so schnell etwas ändern, weil keine Alternativen in Sicht sind.
Im Grunde verfügen alle vorgestellten SOAP Kommunikationsplattformen über den gleichen Grundstock an SOAP Features. Erweiterungen, die über den aktuellen Standard hinausgehen, sind meist auf eine einzelne Plattform beschränkt und lassen sich deshalb nur unter Verzicht auf Plattformunabhängigkeit einsetzen ([46]).

5.4 Sicherheit von SOAP Nachrichten

Sicherheit ist und bleibt ein heiß diskutiertes Thema ([14]). Bei einer Umfrage des World Wide Web Konsortiums unter Serviceentwicklern nach dem SOAP Feature, das sie sich am meisten ersehnen, gab ein Großteil der Befragten an, dass sie sehr für eine standardisierte Umsetzung von Sicherheitsmaßnahmen wären ([17]). Man sieht also, dass Sicherheit von Services ein allgemeiner Wunsch ist. Sieht man sich die SOAP Spezifikation an, so erkennt man, dass SOAP durch seine Konzeption sehr viel Potential in diesem Bereich hat. Man denke nur an die Erweiterbarkeit einer SOAP Nachricht mittels der SOAP Header.
Ein weiteres Ziel dieser Arbeit war es, einen Überblick darüber zu geben, welche Sicherheitskonzepte in den einzelnen untersuchten SOAP Kommunikationsplattformen ihre Umsetzung gefunden haben. Das Gewicht der Untersuchung von Sicherheitskonzepten lag sowohl auf Server- als auch auf Clientseite. Etwaige Konzepte, die in einer Serveranwendung, welche einen Dienst zur Verfügung stellt, umgesetzt werden, müssen ebenfalls auf der Clientseite nachgezogen werden, damit beide Seiten von einem gemeinsamen Grundstock ausgehen können. Darin liegt aber auch der wunde Punkt. Es müssen sowohl der Server als auch der Client zusätzliche Software installieren, um an eine gewisse Basisfunktionalität bezüglich Sicherheit zu kommen. Dies ist nämlich der Fall, weil bei den meisten Implementierungen Sicherheit nur als ein Zusatz gesehen wird und erst mit zusätzlichem Aufwand integriert wird. Im Sinne einer ganzheitlichen Lösung sollte aber ein anderer Ansatz verfolgt werden, der Sicherheit als eine Basisfunktion einer SOAP Implementierung ansieht. An diesem Punkt muss in allen vorgestellten SOAP Kommunikationsplattformen noch gearbeitet werden.

5.5 Schlussresümee

Von den untersuchten SOAP Kommunikationsplattformen sticht vor allem das Apache Axis Projekt heraus. Diese SOAP Implementierung konnte durch ihre Transparenz punkten. Außerdem waren hier die Entwickler am Innovativsten. Die Anzahl der umgesetzten SOAP Features ist bei diesem Projekt besonders hoch. Dies hängt wahrscheinlich auch mit dem Umstand zusammen, dass es sich hierbei um ein Open Source Projekt handelt. Die Umsetzung von neuen Ideen erfolgt besonders schnell und wird oft von mehreren Entwicklern betreut. Ein Schwachpunkt dieser SOAP Implementierung ist aber die Dokumentation. Viele nützliche Eigenschaften lassen sich erst durch lange und mühselige Suche im WWW finden. Eine gut Beschreibung dieser SOAP Kommunikationsplattform findet sich im eben neu erschienenen Buch von Thomas Bayer ([11]). Dieses Buch stellt eine unverzichtbare Lektüre für jeden Serviceentwickler dar, der auf Apache Axis setzt.
Die JAX-RPC Referenzimplementierung von Sun ist ein solides Produkt zur Erstellung von Web Services. Für darüber hinausgehende Experimente ist diese SOAP Implementierung nicht zu haben. Es wird dort sehr stark darauf Bedacht genommen, möglichst allen Profilen des WS-I zu gehorchen. Dies ist vor allem für Entwickler im E-Commerce Bereich sehr von Vorteil. Durch diese Rückversicherung lassen sich die erstellten Produkte leichter verkaufen. An sich ist aber die Idee eines einheitlichen APIs für die Erstellung beziehungsweise Konsumierung eines SOAP Services eine begrüßenswerte Tat.
Im .NET Framework ist die Kommunikation mittels SOAP ein elementarer Bestandteil. Leider verfügt die Basisinstallation des Frameworks nur über eingeschränkte Möglichkeiten, SOAP Features zur Verfügung zu stellen. Erst durch die zusätzliche Installation des Web Service Enhancement Pakets ist eine kommerzielle Nutzung dieser Technologie anzuraten. Die .NET Remoting Architektur bietet eigentlich keine wirkliche Alternative bei der Erstellung von Services, da es einzig und allein für den Austausch von SOAP Nachrichten zwischen .NET Remoting Instanzen konzipiert wurde.
Die untersuchten SOAP Kommunikationsplattformen haben eine bestimmte Kernfunktionalität gemeinsam. Die zusätzlichen Funktionen der einzelnen SOAP Kommunikationsplattformen unterschieden sich dann aber doch erheblich zwischen den einzelnen Implementierungen. Auf den ersten Blick mögen sie schon gleiche SOAP Features zur Verfügung stellen. Die eingesetzten Techniken differieren dann aber doch so eindeutig, dass eine interoperable Nutzung kaum durchführbar ist. Möchte man als Serviceentwickler einen möglichst interoperablen Dienst implementieren, so ist man bestens beraten, wenn man auf die Zusatzfunktionalitäten der jeweiligen SOAP Implementierungen verzichtet, da eine Nutzung meist zu Problemen führt, wenn das

Service über Plattformgrenzen hinweg eingesetzt wird.

Literatur

[1] Conan C. Albrecht. *How clean is the future of SOAP? COMMUNICATIONS OF THE ACM*, 47(2), Februar 2004.

[2] Ross Anderson. *Security Engineering*. Wiley Computer Publishing, erste Auflage, 2001.

[3] Apache. *Apache Axis*, 2003. URL: `http://ws.apache.org/axis/`.

[4] Apache. *Apache Axis C++*, 2004. URL: `http://ws.apache.org/axis/cpp/documentation.html`.

[5] Apache. *Apache WSS4J*, 2004. URL: `http://ws.apache.org/ws-fx/wss4j/`.

[6] M. Baker und M. Nottingham. *IETF Internet Draft „The 'application/soap+xml' media type"*. Technischer Bericht, IETF, Mai 2003. URL: `http://www.ietf.org/internet-drafts/draft-baker-soap-media-reg-03.txt`.

[7] Keith Ballinger. *.NET Web Services*. Addison Wesley, erste Auflage, 2003.

[8] Abbie Barbir, Martin Gudgin und Michael McIntosh. Basic security profile version 1.0, Mai 2004. URL: `http://www.ws-i.org/Profiles/BasicSecurityProfile-1.0-2004-05-12.html`.

[9] Mark Bartel und et al. *W3C Recommendation „XML-Signature Syntax and Processing"*. Technischer Bericht, WWW Consortium, Februar 2002. URL: `http://www.w3.org/TR/2002/REC-xmldsig-core-20020212/`.

[10] John J. Barton, Satish Thatte und Henrik Frystyk Nielsen. *W3C Note SOAP Messages with Attachments*. Technischer Bericht, WWW Consortium, Dezember 2000. URL: `http://www.w3.org/TR/SOAP-attachments/`.

[11] Thomas Bayer, Thilo Frotscher und Marc Teufel. *Java Web Services mit Apache Axis*. Software und Support Verlag, erste Auflage, 2004.

[12] Paul V. Biron und Ashok Malhotra. *W3C Recommendation: XML Schema Part 2: Datatypes*. Technischer Bericht, WWW Consortium, Mai 2001. URL: `http://www.w3.org/TR/2001/REC-xmlschema-2-20010502/`.

[13] Tim Bray, Jean Paoli, C. M. Sperberg-McQueen und Eve Maler. *W3C Recommandation „Extensible Markup Language (XML) 1.0 (Second Edition)“*. Technischer Bericht, WWW Consortium, Oktober 2000. URL: `http://www.w3.org/TR/2000/REC-xml-20001006/`.

[14] L. Buda, J. Grünbauer, J. Jürgens, T. Kuhn und S. Gerbich. *(Un)-Sicherheit bremst Webservices. Information Week*, 5(6), Mai 2004.

[15] J. M. Bull, L. A. Smith, L. Pottage und R. Freeman. Benchmarking java against c and fortran for scientific applications. In *Proceedings of the 2001 joint ACM-ISCOPE conference on Java Grande*. Juni 2001.

[16] Mike Burner. *The Deliberate Revolution. Queue*, 1(1), März 2003.

[17] David A. Chappell und Tyler Jewell. *Java Web Services*. O'Reilly & Associates, Inc., erste Auflage, 2002.

[18] John Cowan und Richard Tobin. *W3C Recommendation XML Information Set*. Technischer Bericht, WWW Consortium, Oktober 2001. URL: `http://www.w3.org/TR/xml-infoset/`.

[19] Francisco Curbera, Matthew Duftler, Rania Khalaf, William Nagy, Nirmal Mukhi und Sanjiva Weerawarana. *Unraveling the Web Services Web: An Introduction to SOAP, WSDL, and UDDI. IEEE Internet Computing*, 6(2), April 2002.

[20] Alexander Davis und Du Zhang. A comparative study of dcom and soap. In *Proceedings of the IEEE Fourth International Symposium on Multimedia Software Engineering*. Dezember 2002.

[21] Dan Davis und Manish Parashar. Latency performance of soap implementations. In *Proceedings of the 2nd IEEE/ACM International Symposium on Cluster Computing and the Grid*. Mai 2002.

[22] Priya Dhawan. *Performance Comparison: .NET Remoting vs. ASP.NET Web Services. Microsoft Developer Network*, September 2002. URL: `http://msdn.microsoft.com/library/en-us/dnbda/html/bdadotnetarch14.asp`.

[23] Andrew Banks et al. Httpr specification. URL: `http://www-106.ibm.com/developerworks/webservices/library/ws-httprspec/`, April 2002.

[24] B. Atkinson et al. *Web Services Security (WS-Security).* Technischer Bericht, Microsoft Developer Network, April 2002. URL: http://msdn.microsoft.com/ws/2002/04/Security/.

[25] Don Box et al. *W3C Note: Simple Object Access Protocol (SOAP) 1.1.* Technischer Bericht, WWW Consortium, Mai 2000. URL: http://www.w3.org/TR/2000/NOTE-SOAP-20000508.

[26] Highland Mary Mountain et al. *W3C Note Web Service Description Language (WSDL) 1.1.* Technischer Bericht, WWW Consortium, März 2001. URL: http://www.w3.org/TR/wsdl.html.

[27] Highland Mary Mountain et al. *W3C Note SOAP Version 1.2 Email Binding.* Technischer Bericht, WWW Consortium, Juni 2002. URL: http://www.w3.org/TR/2002/NOTE-soap12-email-20020626/.

[28] Jonathan Wanagel et al. Building interoperable web services: Ws-i basic profile 1.0. URL: http://msdn.microsoft.com/webservices/building/frameworkandstudio/designing/default.aspx?pull=/library/en-us/dnsvcinter/html/wsi-bp_msdn_landingpage.asp, August 2003.

[29] Keith Ballinger et al. Basic profile version 1.0. URL: http://www.ws-i.org/Profiles/BasicProfile-1.0-2004-04-16.html, April 2004.

[30] Tim Ewald. *The Argument Against SOAP Encoding. Microsoft Developer Network*, Oktober 2002. URL: http://msdn.microsoft.com/library/en-us/dnsoap/html/argsoape.asp?frame=true.

[31] Chris Ferris. *First look at the WS-I Basic Profile 1.0. IBM developerWorks*, Oktober 2002. URL: http://www-106.ibm.com/developerworks/webservices/library/ws-basicprof.html.

[32] Chris Ferris, Anish Karmarkar und Canyang Kevin Liu. *Attachments Profile Version 1.0.* Technischer Bericht, WS-I, August 2004. URL: http://www.ws-i.org/Profiles/AttachmentsProfile-1.0-2004-08-24.html.

[33] Christopher Ferris und Joel Farrell. *What Are Web Services? COMMUNICATIONS OF THE ACM*, 46(6), Juni 2003.

[34] R. Fielding, J.Gettys, J. C. Mogul, H. Frystyk und T. Berners-Lee. *RFC 2616: Hypertext Transfer Protocol HTTP/1.1.* Technischer Bericht, IETF, Januar 1997. URL: http://www.ietf.org/rfc/rfc2616.txt.

[35] N. Freed und N. Borenstein. *Multipurpose Internet Mail Extensions (MIME) Part One*. Technischer Bericht, IETF, November 1996. URL: `http://www.ietf.org/rfc/rfc2045.txt`.

[36] Irini Fundulaki und Arnaud Sahuguet. „share your data, keep your secrets.“. In *Proceedings of the 2004 ACM SIGMOD international conference on Management of data*. Juni 2004.

[37] Madhusudhan Govindaraju, Aleksander Slominski, Venkatesh Choppella, Randall Bramley und Dennis Gannon. Requirements for and evaluation of rmi protocols for scientific computing. In *Proceedings of the 2000 ACM/IEEE conference on Supercomputing*. November 2000.

[38] Martin Gudgin. Secure, reliable, transacted; innovation in web services architecture. In *Proceedings of the 2004 ACM SIGMOD international conference on Management of data*. Juni 2004.

[39] Martin Gudgin, Marc Hadley und Noah Mendelsohn. *W3C Recommendation SOAP Version 1.2 Part 1: Messaging Framework*. Technischer Bericht, WWW Consortium, Juni 2003. URL: `http://www.w3.org/TR/2003/REC-soap12-part1-20030624`.

[40] Martin Gudgin, Mark Hadley und Noah Mendelsohn. *W3C Recommendation SOAP Version 1.2 Part 2: Adjuncts*. Technischer Bericht, WWW Consortium, Juni 2003. URL: `http://www.w3.org/TR/2003/REC-soap12-part2-20030624`.

[41] Martin Gudgin, Noah Mendelsohn, Mark Nottingham und Hervé Ruellan. *W3C Candidate Recommendation „SOAP Message Transmission Optimization Mechanism“*. Technischer Bericht, WWW Consortium, August 2004. URL: `http://www.w3.org/TR/2004/CR-soap12-mtom-20040826/`.

[42] Satoshi Hada und Hiroshi Maruyama. Session authentication protocol for web services. In *Proceedings of the 2002 Symposium on Applications and the Internet*. Januar 2002.

[43] Marc Hadley. *What's New in SOAP 1.2*. Technischer Bericht, XML Europe 2002, Mai 2002. URL: `http://www.idealliance.org/papers/xmle02/dx_xmle02/papers/02-02-02/02-02-02.html`.

[44] Takeshi Imamura, Blair Dillaway und Ed Simon. *W3C Recommendation „XML Encryption Syntax and Processing“*. Technischer Bericht,

WWW Consortium, Dezember 2002. URL: http://www.w3.org/TR/2002/REC-xmlenc-core-20021210/.

[45] Joshy Joseph. *A developer's introduction to JAX-RPC, Part 1.* Technischer Bericht, IBM developerWorks, November 2002. URL: http://www-106.ibm.com/developerworks/library/ws-jaxrpc1/.

[46] A. Klein. *Web-Services verbinden Java und .NET. Bordmittel nur für Basisdienste geeignet. Computerwoche*, 30(35), September 2003.

[47] Mark Little. Models for web sevices tansactions. In *Proceedings of the 2004 ACM SIGMOD international conference on Management of data.* Juni 2004.

[48] Microsoft. *Data Type Support between XML Schema (XSD) Types and .NET Framework Types*, 2004. URL: http://msdn.microsoft.com/library/default.asp?url=/library/en-us/cpguide/html/cpconDataTypeSupportBetweenXSDTypesNETFrameworkTypes.asp.

[49] Microsoft. *.NET 2.0 Whidbey*, 2004. URL: http://msdn.microsoft.com/asp.net/whidbey/default.aspx.

[50] Sun Microsystems. *Java API for XML-based RPC (JAX-RPC 1.1).* Technischer Bericht, JSR-101 Expert Group, Oktober 2003. URL: http://java.sun.com/xml/jaxrpc/index.jsp.

[51] Nilo Mitra. *W3C Proposed Recommendation: SOAP Version 1.2 Part 0: Primer.* Technischer Bericht, WWW Consortium, Mai 2003. URL: http://www.w3.org/TR/2003/REC-soap12-part0-20030507.

[52] Henrik Frystyk Nielsen, Erik Christensen und Joel Farrell. *WS-Attachments.* Technischer Bericht, Internet Draft, Juni 2002. URL: http://www.ibm.com/developerworks/ws-attach.html.

[53] Henrik Frystyk Nielsen und Hervé Ruellan. *W3C Working Draft „SOAP 1.2 Attachement Feature".* Technischer Bericht, WWW Consortium, September 2002. URL: http://www.w3.org/TR/soap12-af/.

[54] Sanjay Patil und Eric Newcomer. *ebXML and Web Services. IEEE Internet Computing*, 7(3), Juni 2003.

[55] Daniel J. Polivy und Roberto Tamassia. Authenticating distributed data using web services and xml signatures. In *Proceedings of the 2002 ACM workshop on XML security.* November 2002.

[56] R. Polster. *IT-Sicherheit auf dem Prüfstand - Neue Herausforderungen durch Web-Services. Objekt Spektrum*, 2003(5), September 2003.

[57] Matt Powell. *Web Services, Opaque Data, and the Attachments Problem. Microsoft Developer Network*, Juni 2004. URL: `http://msdn.microsoft.com/library/default.asp?url=/library/en-us/dnwebsrv/html/opaquedata.asp`.

[58] Sam Ruby. To inifinity and beyond - the quest for soap interoperability. URL: `http://www.interwingly.net/stories/2002/02/01/toInfinityAndBeyoundTheQuestForSoapInteroperability.html`, Februar 2002.

[59] Rich Salz. *Introducing WS-I and the Basic Profile. webservices.xml.com*, März 2003. URL: `http://webservices.xml.com/pub/a/ws/2003/03/04/endpoints.html`.

[60] Yasser Shohoud. *RPC/Literal and Freedom of Choice. Microsoft Developer Network*, April 2003. URL: `http://msdn.microsoft.com/library/en-us/dnwebsrv/html/rpc_literal.asp?frame=true`.

[61] Richard A. Sitze. *Extend JAX-RPC Web services using SOAP headers. IBM developerWorks*, April 2004. URL: `http://www-106.ibm.com/developerworks/webservices/library/ws-tip-extend/`.

[62] Andre Tost. *Using SOAP headers with JAX-RPC. IBM developerWorks*, Oktober 2003. URL: `http://www-106.ibm.com/developerworks/webservices/library/ws-tipjax1.html`.

[63] Christian Weyer. *XML Web Service-Anwendungen mit Microsoft .NET*. Addison-Wesley, erste Auflage, 2002.

[64] Joseph Williams. *J2EE vs. .NET. COMMUNICATIONS OF THE ACM*, 46(6), Juni 2003.