DIPLOMARBEIT

Performanceanalyse von SOAP- und REST- basierten Services in einer Linguistic Resources Umgebung

ausgeführt an der Abteilung für Automatische Sprachverarbeitung der Universität Leipzig

Betreuer:
Prof. Dr. Gerhard Heyer

bearbeitet durch
Sebastian Sander
Harnackstraße 3
04317 Leipzig

Leipzig, 8. Oktober 2010
Zusammenfassung


# Inhaltsverzeichnis

Abkürzungsverzeichnis vi
Abbildungsverzeichnis vii
Tabellenverzeichnis viii

## 1 Einleitung
1.1 Motivation ................................................. 1
1.2 Ziel der Arbeit ............................................. 2
1.3 Arbeitsaufbau .............................................. 2
1.4 Allgemeine und technische Rahmenbedingungen ................. 3

## 2 Web Service Techniken
2.1 Grundlagen ................................................. 5
2.2 Einführung SOAP ........................................... 7
  2.2.1 Spezifikation von SOAP ................................ 8
  2.2.2 Aufbau von SOAP-Nachrichten ......................... 9
  2.2.3 SOAP Faults ........................................... 9
  2.2.4 SOAP-Datentypen und -Encoding ....................... 10
  2.2.5 Nachrichtenaustausch in SOAP ......................... 12
  2.2.6 WSDL ................................................ 12
  2.2.7 Erweiterungen von SOAP Services ..................... 14
2.3 Einführung REST .......................................... 17
  2.3.1 Einschränkungen des REST-Architekturstils ............ 17
  2.3.2 Grundelemente von REST ............................... 19
  2.3.3 Prinzipien einer Ressourcen-orientierten Architektur .... 21
  2.3.4 Erweiterungen von RESTful Services .................. 24
2.4 Performanceevaluation von SOAP- und RESTful-Ser-vices .... 25
  2.4.1 Service-Implementierung ............................... 25
  2.4.2 Versuchsergebnisse .................................. 28
2.5 Zusammenfassung .......................................... 30

## 3 Web Service Frameworks
3.1 Apache Axis 1 ............................................. 31
  3.1.1 Architektur ........................................... 31
  3.1.2 Data Binding ......................................... 34
  3.1.3 Features und Erweiterungen von Apache Axis 1 .......... 35
3.2 Apache Axis 2 ............................................. 37
  3.2.1 Architektur von Apache Axis 2 ......................... 37
  3.2.2 Data Binding ......................................... 42
3.2.3 Features und Erweiterungen von Apache Axis 2 . . . . . . . . . . . 43
3.3 Metro 2.0 . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . 46
  3.3.1 Architektur . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . 46
  3.3.2 Data Binding . . . . . . . . . . . . . . . . . . . . . . . . . . . . . 48
  3.3.3 Features von Metro 2.0 . . . . . . . . . . . . . . . . . . . . . . . . 49
3.4 Performanceevaluation der Web Service Frameworks . . . . . . . . . . . 53
  3.4.1 Service-Implementierung . . . . . . . . . . . . . . . . . . . . . . . 54
  3.4.2 Versuchsergebnisse . . . . . . . . . . . . . . . . . . . . . . . . . . 55
3.5 Zusammenfassung . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . 61

4 Programmierschnittstellen für XML 63
  4.1 Document Object Model . . . . . . . . . . . . . . . . . . . . . . . . . . 63
    4.1.1 Standarisierungen von DOM . . . . . . . . . . . . . . . . . . . . . 63
    4.1.2 DOM Schnittstellen und Knotentypen . . . . . . . . . . . . . . . . 64
    4.1.3 Implementierungen von DOM . . . . . . . . . . . . . . . . . . . . . 66
  4.2 Streaming API for XML . . . . . . . . . . . . . . . . . . . . . . . . . . 67
    4.2.1 Pull Parsing und Push Parsing im Vergleich . . . . . . . . . . . . . 68
    4.2.2 StAX-API . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . 68
    4.2.3 Anwendungsgebiete von StAX . . . . . . . . . . . . . . . . . . . . 70
    4.2.4 StAX Implementierungen . . . . . . . . . . . . . . . . . . . . . . . 71
  4.3 Performanceevaluation der XML-Schnittstellen . . . . . . . . . . . . . . 72
  4.4 Zusammenfassung . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . 73

5 XML Repräsentationen 74
  5.1 Arten von Markup . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . 74
    5.1.1 Inline Markup . . . . . . . . . . . . . . . . . . . . . . . . . . . . . 74
    5.1.2 Stand-Off Markup . . . . . . . . . . . . . . . . . . . . . . . . . . . 75
    5.1.3 Performanceevaluation der Markup-Arten Inline und Stand-Off . . . 76
  5.2 Repräsentation von XML mittels TEI . . . . . . . . . . . . . . . . . . . . . 82
    5.2.1 Einführung TEI . . . . . . . . . . . . . . . . . . . . . . . . . . . . 83
    5.2.2 Markup-Arten in TEI . . . . . . . . . . . . . . . . . . . . . . . . . . 86
    5.2.3 Performanceevaluation der Markup-Arten in TEI . . . . . . . . . . . 86
    5.2.4 Versuchsergebnisse . . . . . . . . . . . . . . . . . . . . . . . . . . 86
  5.3 Zusammenfassung . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . 87

6 Fazit 89
A Anhang 91
  A WSDL Dateien . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . 91
    A.1 Apache Axis 1 . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . 91
    A.2 Apache Axis 2 . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . 92
    A.3 Metro 2.0 . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . 96
  B SOAP-Nachrichten . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . 98
    B.1 Apache Axis 1 . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . 98
    B.2 Apache Axis 2 . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . 99
    B.3 Metro 2.0 . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . 99
  C Markup-Arten für TEI Dokumente . . . . . . . . . . . . . . . . . . . . . . . 100
    C.1 TEI konformes Beispieldokument mit Inline-Markup . . . . . . . . 100
    C.2 TEI konformes Beispieldokument mit Stand-Off-Markup . . . . . . 101
Abkürzungsverzeichnis

AAR  Axis Archive
ACK  Acknowledgement
ADB  Axis Data Binding
API  Application Programming Interface
AVI  Audio Video Interleave
Apache Axis  Apache eXtensible Interaction System
AXIOM  Axis Object Model
CLARIN  Common Language Resources and Technology Infrastructure
CPU  Central Processing Unit
DIME  Direct Internet Message Encapsulation
DNS  Domain Name System
DOM  Document Object Model
DOM4J  Document Object Model for Java
D-SPIN  Deutsche Sprachressourcen-Infrastruktur
EJB  Enterprise JavaBeans
HTML  Hypertext Markup Language
HTTP  Hypertext Transfer Protocol
HTTPS  HyperText Transfer Protocol Secure
IDL  Interface Definition Language
IP  Internetprotokoll
JAR  Java Archive
Java EE 5  Java Platform Enterprise Edition 5
JAX-RPC  Java API for XML-based RPC
JAXB  Java Architecture for XML Binding
JAXP  Java API for XML Processing
JAX-RS  Java API for RESTful Web Services
JAX-WS  Java API for XML Web Services
JDOM  Java Document Object Model
JWS  Java Web Service
MIME  Multipurpose Internet Mail Extensions
MP3  MPEG-1 Audio Layer 3
MTOM  Message Transmission Optimization Mechanism
NACK  Negative Acknowledgement
OASIS  Organization for the Advancement of Structured Information Standards
OMG  Object Management Group
REST  REpresantional State Transfer
ROA  Resource Oriented Architecture
SAAJ  SOAP with Attachments API for Java
SAX  Simple API for XML
SGML  Standard Generalized Markup Language
SJSXP  Sun Java Streaming XML Parser
SOAP4J  SOAP for Java
SSL  Secure Socket Layers
StAX  Streaming API for XML
StAX RI  Streaming API for XML Reference Implementation
SVG  Scalable Vector Graphics
SwA  SOAP with Attachments
TEI  Text Encoding Initiative
UDDI  Universal Description, Discovery and Integration
URI  Uniform Resource Identifier
URL  Uniform Resource Locator
W3C  World Wide Web Consortium
WADL  Web Application Description Language
WAR  Web Application Archive
WCF  Windows Communication Foundation
WSDD  Web Service Deployment Descriptor
WSDL  Web Service Description Language
WSIT  Web Services Interoperability Technology
XML  Extensible Markup Language
XOM  XML Object Model
XOP  XML-binary Optimized Packaging
XSD  XML Schema
XSLT  Extensible Stylesheet Language Transformation
Abbildungsverzeichnis

<table>
<thead>
<tr>
<th>Nummer</th>
<th>Titel</th>
<th>Seitenzahl</th>
</tr>
</thead>
<tbody>
<tr>
<td>1.1</td>
<td>Die Testumgebung</td>
<td>3</td>
</tr>
<tr>
<td>2.1</td>
<td>Web Service Stack</td>
<td>6</td>
</tr>
<tr>
<td>2.2</td>
<td>OSI/ISO Referenzmodell</td>
<td>7</td>
</tr>
<tr>
<td>2.3</td>
<td>Struktur von SOAP-Nachrichten</td>
<td>9</td>
</tr>
<tr>
<td>2.4</td>
<td>Durchschnittliche Gesamtverarbeitungszeit pro Element</td>
<td>29</td>
</tr>
<tr>
<td>3.1</td>
<td>Verarbeitung von SOAP-Nachrichten in Apache Axis 1 auf der Serverseite</td>
<td>32</td>
</tr>
<tr>
<td>3.2</td>
<td>Verarbeitung von SOAP-Nachrichten in Apache Axis 1 auf der Clientseite</td>
<td>33</td>
</tr>
<tr>
<td>3.3</td>
<td>SOAP Elemente in Apache Axis 1</td>
<td>34</td>
</tr>
<tr>
<td>3.4</td>
<td>Beziehung zwischen Context- und Description-Hierarchie</td>
<td>39</td>
</tr>
<tr>
<td>3.5</td>
<td>Verarbeitung von SOAP-Nachrichten in Apache Axis 2</td>
<td>42</td>
</tr>
<tr>
<td>3.6</td>
<td>Metro Web Services Stack</td>
<td>46</td>
</tr>
<tr>
<td>3.7</td>
<td>WSIT Web Service Features</td>
<td>51</td>
</tr>
<tr>
<td>3.8</td>
<td>Klassendiagramm eines Services und einer Ressource</td>
<td>55</td>
</tr>
<tr>
<td>5.1</td>
<td>Aufruf einer Kette von Services durch den Client</td>
<td>77</td>
</tr>
<tr>
<td>5.2</td>
<td>Durchschnittliche Nachrichtengröße der Markup-Arten pro Element</td>
<td>79</td>
</tr>
<tr>
<td>5.3</td>
<td>Durchschnittliche Gesamtverarbeitungszeit der Markup-Arten pro Element</td>
<td>80</td>
</tr>
<tr>
<td>5.4</td>
<td>Durchschnittliche Nachrichtengröße pro Element</td>
<td>87</td>
</tr>
<tr>
<td>5.5</td>
<td>Durchschnittliche Gesamtverarbeitungszeit pro Element</td>
<td>88</td>
</tr>
</tbody>
</table>
# Tabellenverzeichnis

2.1 SOAP-Dokumente für die Spezifikation .......................... 8  
2.2 einfache Datentypen in SOAP ...................................... 11  
2.3 Datenelemente in REST ............................................ 19  
2.4 exemplarische Ressourcen einer linguistischen Umgebung .... 22  
2.5 HTTP-Methoden ...................................................... 22  
2.6 exemplarische Methoden der Ressourcen einer linguistischen Umgebung ...... 23  
2.7 Zusammenfassung der Unterschiede zwischen SOAP und REST .......... 27  
2.8 Versuchsergebnisse von SOAP ..................................... 28  
2.9 Versuchsergebnisse von REST ..................................... 29  
3.1 Kernmodule von Apache Axis 2 ..................................... 38  
3.2 Zusatzmodule von Apache Axis 2 .................................. 38  
3.3 Information Model Hierarchien in Apache Axis 2 ................. 39  
3.4 Phasen einer eingehenden Nachricht in Apache Axis 2 ........... 42  
3.5 Phasen einer ausgehenden Nachricht in Apache Axis 2 .......... 43  
3.6 Zur Verfügung stehende Data Binding Frameworks in Apache Axis 2 . 43  
3.7 Hauptmodule von JAX-WS .......................................... 48  
3.8 Hauptmodule von JAXB ............................................. 49  
3.9 Zusammenfassung der Features der Frameworks .................. 53  
3.10 Versuchsergebnisse der Loadtests von Apache Axis 1 ............. 56  
3.11 Versuchsergebnisse der Loadtests von Apache Axis 2 ............. 56  
3.12 Versuchsergebnisse der Loadtests von Metro 2.0 .................. 56  
3.13 Statistiken der Web Service Kernel beim Stresstest (1) .......... 58  
3.14 Statistiken der Web Service Kernel beim Stresstest (2) .......... 59  
3.15 Zusammenfassung der Rampentestergebnisse ..................... 60  
3.16 Versuchsergebnisse der Loadtests von SOAP und REST in Apache Axis 2 ...... 60  
3.17 Versuchsergebnisse der Loadtests von SOAP und REST in Metro 2.0 .... 61  
4.1 JDOM-Komponenten .................................................. 67  
4.2 XML-Schnittstellen Verarbeitungszeiten auf dem Server .......... 72  
4.3 XML-Schnittstellen Gesamtverarbeitungszeiten ................... 73  
5.1 Nachrichtengröße der Markup-Arten ............................. 79  
5.2 Gesamtverarbeitungszeit der Markup-Arten ....................... 80  
5.3 Zusammenfassung der Inline Markup Parsingergebnisse .......... 81  
5.4 Module in TEI P5 .................................................. 83  
5.5 Versuchsergebnisse der Loadtests von TEI mit Inline-Markup ...... 86  
5.6 Versuchsergebnisse der Loadtests von TEI mit Stand-Off-Markup .... 87
1 Einleitung

1.1 Motivation


Dabei zeichnet sich der bisherige Erstellungsprozess des SOAP-Kernels folgendermaßen ab: ein Drittel bis die Hälfte aller Dateien werden automatisch generiert. Dazu existieren fertige Klassen auf dem Server, um die Services zu generieren. Für den Prozess der Erstellung der Services ist eine Datenbank auf dem Server installiert die Informationen, wie z.B. Name und Typ, zu den zu generierenden Services enthält. Der Typ des Service definiert sich durch seine Arbeitsweise. Ein Service vom Typ Select würde folglich eine einfache Abfrage auf der Datenbank ausführen und dem Client zurückliefern. Der Vorteil dieses Erstellungsprozesses ist, dass jeder Web Service nur ein einziger Eintrag in der Datenbank ist. Sind die Services schließlich generiert, werden die Quellcode-Dateien, die die Implementierung und die Logik der Services beinhalten, schließlich an die richtige Position kopiert und auf dem Server veröffentlicht. Für das Veröffentlichen der Services wird der Server derzeitig immer wieder neu gestartet, was ineffektiv ist. Darüber hinaus wird zusätzlich ein Client für die erzeugten Services und ein Frontend generiert. Das Frontend bietet dann die Möglichkeit, über eine vorherige Anmeldung die generierten Services zu verwenden und zu testen. Da nicht jeder Client fähig ist SOAP-Nachrichten zu verarbeiten, soll der Kernel um die Funktionalität erweitert werden, sowohl SOAP basierte- als auch RESTful Services zur Verfügung zu stellen.


---

\(^1\)Siehe http://www.eaqua.net/.

\(^2\)Im weiteren Verlauf dieser Arbeit wird diese Architektur als SOAP-Kernel bezeichnet.
KAPITEL 1. EINLEITUNG

1.2 Ziel der Arbeit


Da der SOAP-Kernel auf Grundlage von Apache Axis 1 entwickelt wurde, soll außerdem evaluiert werden, ob der Kernel auf Basis anderer Frameworks, wie Apache Axis 2 und Metro 2.0, eine Leistungsteigerung erreichen kann.


1.3 Arbeitsaufbau

Diese Diplomarbeit unterteilt sich in vier Hauptabschnitte:

- Untersuchung von SOAP und REST (Kapitel 2)
- Untersuchung ausgewählter Web Service Frameworks (Kapitel 3)
- Untersuchung ausgewählter XML-Programmierschnittstellen (Kapitel 4)
- Untersuchung von XML-Repräsentationen (Kapitel 5)

Im 2. Kapitel wird die Grundlage für die darauffolgenden Kapitel erarbeitet. Hier werden die Web Service Technologien SOAP und REST erläutert. Am Ende dieses Kapitels sind die Performancetests und die dazugehörige Auswertung der Tests der beiden Technologien, SOAP und REST, ersichtlich.

KAPITEL 1. EINLEITUNG


Das Ende dieser Arbeit ist ein Fazit mit einer kritischen Betrachtung dieser.

1.4 Allgemeine und technische Rahmenbedingungen

Dieser Abschnitt gibt einen Überblick über die technischen Rahmenbedingungen wie Server, Netzwerk und Client. Die nachfolgende Abbildung verdeutlicht die Testumgebung.

Abbildung 1.1: Die Testumgebung


Für das Monitoring des Servers wurde JavaMelody3 verwendet. Dies ist ein Tool zur Überwachung von Serveraktivitäten, das unter anderem die Auslastung der CPU, den Speicherverbrauch, die Anzahl der Anfragen pro Minute oder die Systemfehler misst. Die Integration von JavaMelody in die einzelnen Webanwendungen verlief ohne Probleme. Es mussten lediglich zwei Java Archive (JAR)-Dateien in die jeweilige Webanwendung eingebunden und über die Datei web.xml registriert werden. Dabei war es möglich, das Interval der Erzeugung der Messpunkte zu konfigurieren. Es wurden einige Versuche mit verschiedenen Intervallen durchgeführt. Das Resultat daraus war, dass ein minütliches Interval die Testergebnisse lediglich um 0.1% verfälscht. Dies schien geeignet und vernachlässigbar zu sein.

Zur Generierung der Testanfragen wurde JMeter 2.3.44 verwendet. JMeter ist ein Tool zur Generierung von Testplänen jeglicher Art. Für diese Arbeit wurde es verwendet, um eine gewisse Anzahl von Anfragen auf dem Server auszuführen. Es bietet außerdem die Möglich-

3 Siehe (JavaMelody, 2010).
4 Siehe (Project, 2010).
keit, mehrere Clients parallel Anfragen ausführen zu lassen und die Ergebnisse in Dateien zu speichern. Anzumerken ist, dass alle Tests mehrfach und, wo dies möglich war, auf verschiedene Art und Weise durchgeführt wurden, um die Testergebnisse zu verifizieren.

Arten der Testdurchführung:

- **Komplett:**
  - Alle zusammengehörigen Tests wurden ohne Unterbrechung und Neustart des Servers durchgeführt.
  - Beispiel: Der SOAP-Kernell auf Basis von Apache Axis 1 wurde zuerst mit fünf verschiedenen Nachrichtengrößen (1-, 100- 1.000-, 10.000- und 100.000 Elementen), durchgeführt. Anschließend Apache Axis 2 mit den verschiedenen Nachrichtengrößen und danach Metro 2.0. Dieser Testdurchlauf geschah nacheinander und ohne Neustart des Servers.

- **in sich abgeschlossen:**
  - Alle zusammengehörigen Tests wurden ohne Unterbrechung und Neustart des Servers durchgeführt.

- **einzeln:**
  - Die Test wurden für jede Nachrichtengrößen einzeln durchgeführt.
2 Web Service Techniken

2.1 Grundlagen


Gartner Group

“A Web service is a custom end-to-end application that interoperates with other commercial and custom software through a family of XML interfaces (like SOAP, UDDI and WSDL) to perform useful business functions.”2

Frankfurter Allgemeine Zeitung

“Web Services sind Softwarebausteine, die Programme, die auf unterschiedlichen Netzwerkrechnern laufen, über das Internet zu einer Anwendung miteinander verknüpfen.”3

IBM

“Web services are self-describing, self-contained, modular applications that can be published, locatet, and invoked across the Web.”4

Zusammenfassend kann gesagt werden, dass Web Services Programme oder Funktionen sind, die auf entfernten Rechnern liegen und öffentlich, über Intranet bzw. Internet, zugänglich sind. Dabei ist anzumerken, dass SOAP für die Realisierung von Web Services nicht zwingend notwendig ist. Eine weitere weit verbreitete Möglichkeit Services zu entwickeln

ist REST. Diese beiden Technologien werden in den nachfolgenden Abschnitten dieses Kapitels ausführlich erläutert. Die grundlegende technische Architektur von Web Services ist ähnlich dem ISO/OSI-Referenzmodell\(^5\) und besteht aus aufeinander aufbauenden Schichten:

![Web Service Stack Diagram](image)

Abbildung 2.1: Web Service Stack


Im **Description Layer** werden die in den Web Services enthaltenen Methoden vom Service-provider beschrieben, damit der Konsument diese dann aufrufen kann. Für die Beschreibung eines Services werden die Sprachenstandards *Web Service Description Language* (**WSDL**) oder *Web Application Description Language* (**WADL**) genutzt.

Im **Packaging Layer** wird das Protokoll definiert, das verwendet wird, um die Daten auszutauschen\(^8\).

Die beiden untersten Ebenen, **Transport-** und **Network-Layer**, sind elementar und setzen sich aus den bekannten Schichten des ISO/OSI Referenzmodells zusammen, das in der nachfolgenden Abbildung aufgezeigt wird.

---

\(^6\) Siehe http://www.oasis-open.org/committees/ciq/ciq.html
\(^8\) Vgl. (Dörnemann, 2008).

Die Netzwerkschicht in Abbildung 2.1 bildet dabei den physikalischen Übertragungsweg zwischen Servicekonsument und Servicesprovider. Sie ist eine Zusammenfassung der vier unteren Schichten des Referenzmodells. Die Kommunikation erfolgt in der Regel über TCP/IP.

### 2.2 Einführung SOAP


---

9Vgl. (Gustavo Alonso, 2004) S. 152.

2.2.1 Spezifikation von SOAP


<table>
<thead>
<tr>
<th>Dokument</th>
<th>Inhalt</th>
</tr>
</thead>
<tbody>
<tr>
<td>SOAP Version 1.2 Part 0: Primer</td>
<td>Tutorial zu SOAP</td>
</tr>
<tr>
<td>SOAP Version 1.2 Part 1: Messaging Frame-</td>
<td>Kernspezifikation von SOAP</td>
</tr>
<tr>
<td>work</td>
<td></td>
</tr>
<tr>
<td>SOAP Version 1.2 Part 2: Adjuncts</td>
<td>HTML-Dokument, MPEG-1 Audio Layer 3 (MP3)</td>
</tr>
<tr>
<td>Specification Assertions and Test Coll-</td>
<td>Media Type, Last-Modified Time</td>
</tr>
<tr>
<td>lection</td>
<td></td>
</tr>
</tbody>
</table>

Tabelle 2.1: SOAP-Dokumente für die Spezifikation


Das letzte, der hier vorgestellten Dokumente, ist Specification Assertions and Test Collection, welches als Hauptziel die Förderung der Interoperabilität zwischen verschiedenen SOAP 1.2 Implementierungen hat.

In den nachfolgenden Kapiteln werden die kurz erläuterten Spezifikationen im Detail behandelt.

---

11 Vgl. (W3C, 2007)
2.2.2 Aufbau von SOAP-Nachrichten


Abbildung 2.3: Struktur von SOAP-Nachrichten


2.2.3 SOAP Faults

Ein SOAP-Fault ist eine spezielle Art einer SOAP-Nachricht, die Informationen über Fehler beinhaltet, die während der Verarbeitung aufgetreten sind. Sie kann genau einmal im SOAP-Body enthalten sein und gilt als ein spezielles SOAP-Body Element und muss, falls

\[\text{Vgl. (James Snell, 2002) Seite 11 - 19.}\]

Der Fault Code ist ein algorithmisch generierter Wert, um den Fehler zu identifizieren. Dieser enthält einen der Werte:

**VersionMismatch:** Der SOAP-Envelope verwendet einen ungültigen Namespace.

**MustUnderstand:** Ein Element im SOAP-Header mustUnderstand hatte den Wert true, doch der Nachrichtenempfänger konnte diese Element nicht verarbeiten.

**Client/Receiver:** Bei der Formatierung der Nachricht trat ein Fehler auf oder die enthaltenen Daten waren fehlerhaft.

**Server/Sender:** Ein Fehler trat auf der Seite des Serviceprodukten auf.

**DataEncodingUnknown:** Das angegebene SOAP-Encoding wird nicht unterstützt.

Der Fault String ist dabei eine für den Menschen lesbare Nachricht, die den Fehler beschreibt. Der Fault Actor gibt den Knoten an, wo der Fehler auftrat. Im Element Fault Details stehen anwendungsspezifische Details zum Fehlerhergang. 13

```xml
<?xml version='1.0' encoding='UTF-8'?>
<soapenv:Envelope xmlns:soapenv="http://www.w3.org/2003/05/soap-envelope">
  <soapenv:Body>
    <soapenv:Fault>
      <soapenv:Code>
        <soapenv:Value>soapenv:Receiver</soapenv:Value>
      </soapenv:Code>
      <soapenv:Reason>
        <soapenv:Text xml:lang="en-US">Exception occurred while trying to invoke service method getSentences</soapenv:Text>
      </soapenv:Reason>
      <soapenv:Detail />
    </soapenv:Fault>
  </soapenv:Body>
</soapenv:Envelope>
```

Listing 2.1: Beispiel einer SOAP Fehlernachricht

2.2.4 SOAP-Datentypen und -Encoding


<table>
<thead>
<tr>
<th>Datentyp</th>
<th>XSD</th>
<th>Java</th>
</tr>
</thead>
<tbody>
<tr>
<td>Zeichenketten</td>
<td>xsd:string</td>
<td>java.lang.String</td>
</tr>
<tr>
<td>Ganzzahlwerte</td>
<td>xsd:integer</td>
<td>java.math.BigInteger</td>
</tr>
<tr>
<td>Ganzzahlwerte</td>
<td>xsd:int</td>
<td>int</td>
</tr>
<tr>
<td>Ganzzahlwerte</td>
<td>xsd:long</td>
<td>long</td>
</tr>
<tr>
<td>Ganzzahlwerte</td>
<td>xsd:short</td>
<td>short</td>
</tr>
<tr>
<td>Dezimalzahlwerte</td>
<td>xsd:decimal</td>
<td>java.math.BigDecimal</td>
</tr>
<tr>
<td>Gleitkommazahlwerte</td>
<td>xsd:float</td>
<td>float</td>
</tr>
<tr>
<td>Gleitkommazahlwerte</td>
<td>xsd:double</td>
<td>double</td>
</tr>
<tr>
<td>Wahrheitswerte</td>
<td>xsd:boolean</td>
<td>boolean</td>
</tr>
<tr>
<td>Ganzzahlwerte</td>
<td>xsd:byte</td>
<td>byte</td>
</tr>
<tr>
<td>XML qualifizierte Namen</td>
<td>xsd:QName</td>
<td>javax.xml.namespace.QName</td>
</tr>
<tr>
<td>Zeitverwaltung</td>
<td>xsd:dateTime</td>
<td>javax.xml.datatype/XMLGregorianCalendar</td>
</tr>
<tr>
<td>Array aus Bytes</td>
<td>xsd:base64Binary</td>
<td>byte[]</td>
</tr>
<tr>
<td>Array aus Bytes</td>
<td>xsd:hexBinary</td>
<td>byte[]</td>
</tr>
<tr>
<td>Ganzzahlwerte</td>
<td>xsd:unsignedInt</td>
<td>long</td>
</tr>
<tr>
<td>Ganzzahlwerte</td>
<td>xsd:unsignedShort</td>
<td>int</td>
</tr>
<tr>
<td>Ganzzahlwerte</td>
<td>xsd:unsignedByte</td>
<td>short</td>
</tr>
<tr>
<td>Zeitverwaltung</td>
<td>xsd:time</td>
<td>javax.xml.datatype/XMLGregorianCalendar</td>
</tr>
<tr>
<td>Zeitverwaltung</td>
<td>xsd:date</td>
<td>javax.xml.datatype/XMLGregorianCalendar</td>
</tr>
<tr>
<td>Zeichenkette</td>
<td>xsd:anySimpleType</td>
<td>java.lang.String</td>
</tr>
<tr>
<td>Zeitverwaltung</td>
<td>xsd:duration</td>
<td>javax.xml.datatype.Duration</td>
</tr>
<tr>
<td>Namespace</td>
<td>xsd:NOTATION</td>
<td>javax.xml.namespace.QName</td>
</tr>
</tbody>
</table>

Tabelle 2.2: einfache Datentypen in SOAP


einzelnen Elements, da sie alle den gleichen Namen besitzen. Ein dritter, komplexer Typ in SOAP sind die Mischformen aus den Strukturen und Feldern. Dabei können Elemente, wie auch in Strukturen über den Namen referenziert werden, aber auch mehrfach auftauchen, wie in Feldern. Die zusammengesetzten Datentypen können wiederrum ineinander verschachtelt werden.\textsuperscript{14}

### 2.2.5 Nachrichtenaustausch in SOAP


### 2.2.6 WSDL

Da Web Services vom Service Provider beschrieben werden müssen, wird dazu WSDL verwendet. Es basiert auf XML und ist plattformunabhängig. Mit Hilfe von WSDL werden die veröffentlichten Funktionen, Daten, Datentypen und Austauschprotokolle eines Web Services beschrieben. Im Anhang A sind automatisch erzeugte, komplette WSDL-Dateien der im nächsten Kapitel untersuchten Web Service Frameworks aufgezeigt. Im Folgenden werden die Elemente einer Dienstbeschreibung kurz erläutert. Das Element \textit{message} beinhaltet abstrakte Definitionen der Daten, die zwischen Konsument und Provider des Services ausgetauscht werden. Hier wird eine Auflistung der Methoden, die der Web Service veröffentlicht, die dazugehörigen Eingangsparameter sowie das Format der zurückgegebenen Ergebnismenge angegeben.

```xml
<wsdl:message name="getSentencesResponse">
  <wsdl:part name="parameters" element="ns:getSentencesResponse"/>
</wsdl:message>
```

Listing 2.2: Beispiel eines message Elementes


\textsuperscript{15} Siehe http://docs.openlinksw.com/virtuoso/ws-routing.html/.

Im WSDL-Element *types* werden die Datentypen definiert, die in den Nachrichten ausge- 
tauscht werden. Das Typsystem kann verwendet werden, um in einer Nachricht Typen zu 
definieren. Ob das Format tatsächlich XML ist bzw. ob das XML Schema (XSD) das spe- 
zielle Format validiert braucht nicht berücksichtigt werden. Dies erhält seine Wichtigkeit, 
wenn für die gleiche Nachricht mehrere Bindings definiert werden oder wenn nur ein Bin- 
ding existiert, aber der Bindingtyp kein eigenes Typsystem hat. 

Im Element *portType* wird die Serviceschnittstelle und die Serviceoperationen festgelegt. 
Somit wird hier gesagt, über welchen Port bestimmte Methoden des Services vom Client 
augerufen und die Antworten des Providers gesendet werden. Wie im folgenden Listing zu 
sehen ist, referenziert jede in portType enthaltene Operation auf eine Eingangs- und eine 
Ausgangsnachricht.

```
<wsdl:portType name="SentencesPortType">
    <wsdl:operation name="getSentences">
        <wsdl:input message="ns:getSentencesRequest">
            wsaw:Action="urn:getSentences"/>
        <wsdl:output message="ns:getSentencesResponse">
            wsaw:Action="urn:getSentencesResponse"/>
        <wsdl:fault message="ns:RemoteException">
            name="RemoteException">
            wsaw:Action="urn:getSentencesRemoteException"
        </wsdl:fault>
    </wsdl:operation>
</wsdl:portType>
```

Listing 2.3: Beispiel eines portType Elementes

Die Elemente des WSDL-Dokumentes werden in zwei Gruppen untergliedert. Die Elemente 
messages, types und portTypes gehören zu der Gruppe der abstrakten Definitionen. 
Zu der Gruppe der konkreten Definitionen gehören die im folgenden erläuterten Elemente: 
Das Element *binding* spezifiziert das konkret eingesetzte Protokoll und das Datenformat für 
die Operationen, die im ElementportType definiert wurden.

```
<wsdl:binding name="SentencesSoap11Binding">
    <soap:binding transport="http://schemas.xmlsoap.org/soap/http">
        <wsdl:operation name="getSentences">
            <soap:operation style="document">
                soapAction="urn:getSentences" style="document"/>
            <wsdl:input>
                <soap:body use="literal"/>
            </wsdl:input>
            <wsdl:output>
                <soap:body use="literal"/>
            </wsdl:output>
            <wsdl:fault name="RemoteException">
                <soap:fault use="literal"
```

KAPITEL 2. WEB SERVICE TECHNIKEN

Das Element service fasst die Menge der dazugehörigen Ports zusammen. Dieses Element beinhaltet Elemente vom Typ port, in welchen die Adresse des Serviceproviders angegeben wird, unter der der Web Service erreichbar ist. Diese Adresse wird für ein Binding angegeben. Dabei ist zu beachten, dass das Element port nicht mehr als eine Adresse beinhalten darf.

In diesem Beispiel werden drei Ports beschrieben. Der Erste veröffentlicht einen SOAP-Version 1.1-Endpunkt, der Zweite einen SOAP-Version 1.2-Endpunkt und der Dritte einen HTTP-Endpunkt. ¹⁷

2.2.7 Erweiterungen von SOAP Services

Sicherheit


WS-Security verwendet für die Realisierung von Sicherheit bereits bestehende Standards. Dabei kommen Kerberos und X.509 für die Authentifizierung, XML Encryption und XML

¹⁷Vgl. (Erik Christensen, 2001).

Zuverlässige Nachrichtenübertragung


AtLeastOnce: jede Nachricht, die gesendet wird, wird auch ausgeliefert, sonst wird ein Fehler erzeugt. Duplikate werden nicht weiter behandelt, da hier die korrekte Auslieferung mindestens einer Nachricht zu gesichert wird.

ExactlyOnce: Jede Nachricht wird genau einmal gesendet.


Die Kontrolle über die Strategicodes AtMostOnce, AtLeastOnce, ExactlyOnce oder InOrder liegt bei dem Sender. Dieser identifiziert Erfolge und Misserfolge in der Übertragung durch die vom Empfänger geschickten Acknowledgement (ACK), welche die Sequenznummer der gerade empfangenen Nachricht beinhaltet.


19Siehe (Tobias Hauser, 2004) S. 137 - 146.
20Vgl. (Davis, 2005).
21Vgl. (Francisco Curbera, 2004).


**Transaktionen**

Die Spezifikation *WS-Transaction* beschreibt einen Mechanismus, der es erlaubt, Operationen entweder vollständig oder gar nicht auszuführen. Sie besteht aus drei Unterspezifikationen.

**WS-Coordination** stellt Protokolle zur Verfügung, die es erlauben Aktionen von Anwendungen zu koordinieren. Außerdem schreibt WS-Coordination vor, dass ein Koordinator existiert, bei dem sich die Web Services registrieren.


**Nachrichtenoptimierung**


Nachrichtenoptimierung garantiert eine effiziente Übertragung von Nachrichten über das Internet und versetzt einen Serviceendpunkt in die Lage große binäre Nutzdaten zu erkennen.

---

22 Siehe (Schlimmer, 2006).


24 Vgl. (Feingold, 2005) und (IBM, 20004)
diese aus dem SOAP-Body zu entfernen, die Nutzdaten mit einem effektiveren Kodierungsmechanismus zu kodieren und wieder in die SOAP-Nachricht als Attachment hinzuzufügen. Zwei dieser Mechanismen sind:

**SOAP with Attachments (SwA):** ist ein Vorschlag der W3C für den Transport von SOAP-Nachrichten mit Attachments innerhalb von Multipurpose Internet Mail Extensions (MIME) bzw. Direct Internet Message Encapsulation (DIME). In SwA besteht eine SOAP-Nachricht aus zwei Teilen. Dem primären Part SOAP-Envelope und dem sekundären Part Attachment. Der Attachment Part kann über eine eindeutige Content-ID von der SOAP Nachricht referenziert werden.\(^{25}\)


### 2.3 Einführung REST


Fielding betrachtet das moderne Web als eine Instanz des REST Architekturstils. Dabei ist jedoch zu beachten, dass REST keine spezifische Technologie darstellt und nicht an HTTP gebunden ist. Es beschreibt lediglich abstrakte Eigenschaften, die ein REST-konformes System erfüllen muss.


#### 2.3.1 Einschränkungen des REST-Architekturstils

Der REST-Architekturstil ist aus mehreren Einschränkungen zusammengesetzt, denen einer Architektur genügen muss. In den nachfolgenden Unterabschnitten werden diese anhand von


\(^{26}\) Vgl. (Martin Gudgin, 2005a,b; Unbekannt, 2010) und (Thilo Frotscher, 2007) Seite 414 - 419.

(Fielding, 2000) S. 76ff beschrieben.

**Client-Server**

Der Client-Server-Architekturstil ist das erste Constraint, das sich in REST wiederfindet. Darin wird das sogenannte *Separation of Concerns*-Prinzip, welches die Trennung von Zuständigkeiten vorschreibt, verwendet. Somit verbessert sich die Skalierbarkeit des Servers, da die Komponenten einfach gehalten werden können.

**Zustandlosigkeit**


**Cachefähigkeit**


**Einheitliche Schnittstelle**

Das zentrale Feature des REST-Architekturstils ist, dass zwischen Softwarekomponenten eine einheitliche, allgemeine Schnittstelle besteht. Dadurch erhöht sich die Überwachbarkeit der Transaktionen und die Implementierungen werden von deren Services losgelöst. Der Nachteil daraus ist, dass sich aufgrund einer standardisierten Schnittstelle die Effizienz vermindert. Um solch eine einheitliche Schnittstelle realisieren zu können, wurden mit:

- Bestimmung von Ressourcen,
- Manipulation von Ressourcen über Repräsentationen,
- selbstbeschreibende Nachrichten und
- Hypermedia als Engine des Application State

die vier weitere Einschränkungen definiert. Diese werden im nachfolgenden Abschnitt genauer erläutert.

---

Geschichtetes System


Code on Demand

Die letzte Einschränkung im REST-Architekturstil ist Code on Demand. Damit ist gemeint, dass REST es dem Client ermöglicht den Quellcode herunterzuladen sowie auszuführen, welcher in Applets oder Scripten zur Verfügung gestellt wird. Somit wird die Clientfunktionalität zur Laufzeit dynamisch erweitert, was ihn leichtgewichtiger macht.

2.3.2 Grundelemente von REST

Datenelement


<table>
<thead>
<tr>
<th>Datenelement</th>
<th>Beispiele</th>
</tr>
</thead>
<tbody>
<tr>
<td>resource</td>
<td>Ziel einer Hypertext Ressource</td>
</tr>
<tr>
<td>resource identifier</td>
<td>URL,URN</td>
</tr>
<tr>
<td>representation</td>
<td>HTML-Dokument, MP3 Datei</td>
</tr>
<tr>
<td>representation metadata</td>
<td>Media Type, Last-Modified Time</td>
</tr>
<tr>
<td>resource metadata</td>
<td>Kopfzeile eines HTTP-Responses (alternates, vary)</td>
</tr>
<tr>
<td>control data</td>
<td>Kopfzeile eines HTTP-Responses (if-modified-since)</td>
</tr>
</tbody>
</table>

Tabelle 2.3: Datenelemente in REST


Die Komponenten von REST führen Aktionen auf einer Ressource aus. Dies wird realisiert, indem sie eine Repräsentation dieser Ressource verwenden. Somit wird der tatsächliche oder der zukünftige Zustand einer Ressource festgehalten. Die Repräsentationen werden dann an andere Komponenten übertragen und bestehen aus Information aus der Ressource, den Metadaten, die diese Daten beschreiben sowie, auf Veranlassung, aus Metadaten, die

²⁹Aus (Fielding, 2000)Table 5-1. REST Data Elements.
KAPITEL 2. WEB SERVICE TECHNIKEN

20
die Metadaten beschreiben. Metadaten sind dabei sogenannte Namen-Werte-Paare, wobei
der Name einem Standard entspricht, der die Struktur der Werte und die Semantik definiert.
Antwortdokumente können dabei sowohl resources metadata als auch representation metadata beinhalten. Dabei sind resources metadata Informationen über Ressourcen, die nicht einer bestimmten Ressource zugeordnet werden können. Control data definiert den Zweck einer Nachricht, die zwischen den Komponenten ausge-

Konnektoren

Um den Zugriff auf Ressourcen und die Übertragung der Repräsentationen der Ressourcen zu verkapseln, werden in REST verschiedene Typen von Konnektoren verwendet. Konnektoren sind abstrakte Schnittstellen für die Kommunikation zwischen Komponenten. Sie tragen dazu bei, dass der Kommunikationsmechanismus und dessen Implementierung getrennt wer-
den kann.
Alle auf REST basierenden Interaktionen sind zustandslos, was bedeutet, dass der Server lediglich den Zustand seiner eigenen Ressourcen verwaltet und nicht den des Clients und der Anwendung. Der Ansatz der Zustandslosigkeit hat den Nachteil, dass jeder Request alle benötigten Informationen beinhalten muss, damit er erfolgreich abgeschlossen werden kann. Dies kann eine Erhöhung der Netzwerklast mit sich bringen. Die Vorteile, die aus der Zu-
standslosigkeit resultieren, sind dabei von größerer Bedeutung. Die Konnektoren müssen nicht den Zustand der Anwendung zwischen den Requests speichern. Interaktionen können parallel abgearbeitet werden, obwohl zwischen den Anfragen keinerlei Zusammenhang be-
steht.
Mit Client-, Server-, Cache-, Resolver- und Tunnel-Konnektor existieren in REST fünf ver-
schiedene Typen von Konnektoren.
Die beiden Hauptkonnektoren sind der Client- und der Serverkonnektor. Der Unterschied zwischen beiden liegt darin, dass ein Clientkonnektor eine Kommunikation beginnt, indem er ein Request sendet. Der Serverkonnektor hingegen wartet auf Requests und antwortet auf diese.
Der Resolverkonnektor übersetzt, komplett oder teilweise, den Resource Identifier in eine Netzwerkadresse, um die für den physikalischen Netzwerkzugriff benötigten Daten zu er-
halten.
Der Tunnelkonnektor ermöglicht es, Informationen über Systemgrenzen hinweg auszutau-
schen. Der Grund, warum dieser Konnektor nicht als Teil der Netzwerkinfrastruktur ausge-
lagert wurde, ist, dass sich einige REST-Komponenten dynamisch von einer aktiven Kom-
ponente zu einem Tunnel ändern.
Typen von Komponenten


2.3.3 Prinzipien einer Ressourcen-orientierten Architektur


Adressierbarkeit

Ressourcen sind in einer Ressourcen-orientierten Architektur Elemente, die Informationen zur Verfügung stellen. Dabei kann dies ein Algorithmus, eine Datenbankoperation oder Ähnliches sein, die über das Web zugänglich ist. Die Adresse einer Ressource sowie deren Name wird durch die URI festgelegt. Dadurch kann sie global eindeutig identifiziert werden.

**Protokoll: // Host:Port/Pfad?Querystring/#fragment**

Listing 2.6: Aufbau einer URI

KAPITEL 2. WEB SERVICE TECHNIKEN

<table>
<thead>
<tr>
<th>URI</th>
<th>Bedeutung der Ressource</th>
</tr>
</thead>
<tbody>
<tr>
<td>/sentences/getSentences/</td>
<td>Auflistung aller existierenden Sätze innerhalb eines Intervalls</td>
</tr>
<tr>
<td>/sentences/{sentence_id}/getReference/</td>
<td>Liefert Belegstelle für einen bestimmten Satz</td>
</tr>
<tr>
<td>/sentences/{sentence_id}/getCitation/</td>
<td>Liefert Zitate für einen bestimmten Satz</td>
</tr>
<tr>
<td>/words/{word_id}/</td>
<td>Liefert ein bestimmtes Wort</td>
</tr>
<tr>
<td>/words/{word_id}/getFrequencies/</td>
<td>Liefert die Frequenz eines Wortes aus einem Satz</td>
</tr>
<tr>
<td>/words/{word_id}/position/</td>
<td>Liefert die Position eines Wortes in einem Satz</td>
</tr>
<tr>
<td>/words/{word_id}/cooccurrences/</td>
<td>Auflistung der signifikanten Kookurrenzen für ein bestimmtes Wort</td>
</tr>
<tr>
<td>/authors/</td>
<td>Auflistung aller registrierten Autoren</td>
</tr>
<tr>
<td>/authors/{author_id}/</td>
<td>Liefert einen speziellen Autor</td>
</tr>
</tbody>
</table>

Tabelle 2.4: exemplarische Ressourcen einer linguistischen Umgebung

Einheitliche Schnittstelle

Es wird ein kleiner Satz an wohldefinierten Methoden, die es ermöglichen Ressourcen zu manipulieren, eingeführt. Die Idee ist, dass nur die generischen Methoden von HTTP für die RESTful Services genutzt werden. In Tabelle 2.5 ist eine Übersicht dieser HTTP-Methoden dargestellt. Es gibt noch weitere, wie TRACE und CONNECT, jedoch sind diese für RESTful Web Services nicht relevant und werden aus diesem Grund nicht näher betrachtet.

<table>
<thead>
<tr>
<th>Methode</th>
<th>Beschreibung</th>
</tr>
</thead>
<tbody>
<tr>
<td>GET</td>
<td>ist eine read-only Operation. Dient der Abfrage der Information einer Resource.</td>
</tr>
<tr>
<td>PUT</td>
<td>fügt einen Datensatz hinzu oder aktualisiert ihn. Der Client kennt die Identität der Ressource, die hinzufügt oder aktualisiert wird.</td>
</tr>
<tr>
<td>DELETE</td>
<td>löschen einer Ressource</td>
</tr>
<tr>
<td>POST</td>
<td>Hinzufügen von Informationen, in Form einer Repräsentation einer Resource.</td>
</tr>
<tr>
<td>HEAD</td>
<td>ähnelt GET mit der Ausnahme, dass nur Metainformationen einer Resource zurück geliefert werden.</td>
</tr>
<tr>
<td>OPTIONS</td>
<td>wird verwendet, um Informationen über die Kommunikationsoptionen und Ressourcen abzufragen.</td>
</tr>
</tbody>
</table>

Tabelle 2.5: HTTP-Methoden

Die Beschränkung der Methoden hat Vorteile. Es ist bei einer URI, die auf einen Web Service verweist ersichtlich, welche Methoden für diese Ressource verfügbar sind. Außerdem werden keine neuen Client Libraries benötigt, denn die meisten Programmiersprachen stellen eine solche Bibliothek standardmäßig bereit. In der nachfolgenden Tabelle 2.6 sind mögliche Operationen auf den aus Tabelle 2.4 dargestellten Ressourcen aufgelistet.
HTTP Methode | Bedeutung
---|---
GET | Resource /sentences/getSentences/
| Abfrage aller existierenden Sätze innerhalb eines Intervalls
PUT | nicht verwendet
POST | Hinzufügen von Sätzen
DELETE | Löschen von Sätzen innerhalb eines Intervalls
GET | Resource/authors/{author_id}/
| Liefert einen speziellen Autor
PUT | Änderung von Autorendaten
POST | Einen neuen Autor hinzufügen
DELETE | Löschen eines Autors

Tabelle 2.6: exemplarische Methoden der Ressourcen einer linguistischen Umgebung

Zustandslosigkeit


Repräsentationen


Verbund von Ressourcen

Eine Vielzahl von RESTful Services sind Repräsentationen von Hypermedia. Sie können auch Links zu anderen Ressourcen beinhalten. Wenn nun Ressourcen intelligent miteinander verbunden werden, ist es möglich mit nur einem einzigen Link den Einstiegspunkt für weitere Services bereitzustellen. Immer noch ein Beispiel zum Thema Verbund von Ressourcen aus der linguistischen Umgebung zu nennen, könnte ein Aufruf der Ressource /sentences/getSentences/ die Links zu den Wörtern innerhalb des Satzes, also zu der Ressource /words/{word_id}/, sowie zu der Ressource /authors/{author_id}/, die einen bestimmten Autor zurückliefer, beinhalten.

---

30 Siehe (Ruby, 2007) S. 95.
2.3.4 Erweiterungen von RESTful Services

In (Ruby, 2007) werden Möglichkeiten besprochen, wie RESTful Services gesichert werden können. Diese werden in diesem Abschnitt genannt. Dabei wird ein Vergleich zum Abschnitt 2.2.7 gezogen, um aufzuzeigen, wie ein Äquivalent zu den für die SOAP Services existierenden WS-* Spezifikationen geschaffen werden kann.

Sicherheit


Zuverlässige Nachrichtenübertragung


Transaktionen


\(^{31}\)Eine lineare Abbildung von \( p : V \rightarrow V \) mit \( p^2 = p \) wird eine Projektion oder idempotent genannt. Siehe (Pareigis, 2000) S. 176

Dieses Szenarios kann auf Serverseite realisiert werden, indem eine Warteschlange für die zu den Transaktionen gehörigen Aktionen erstellt wird. Sobald die Transaktion bestätigt wird, kann der Server zum Beispiel eine Datenbanktransaktion starten und diese committen. Da Transaktionen dem Konzept von RESTful Services widersprechen, sollte die Verwendung vermieden werden.

2.4 Performanceevaluation von SOAP- und RESTful-Services


2.4.1 Service-Implementierung


33 Diese Tabelle enthält eine Zusammenfassung der erarbeiteten Unterschiede zwischen SOAP und REST. Aus diesem Grund wird von einer erneuten Angabe der Quellen abgesehen.

34 Im weiteren Verlauf dieser Arbeit wird dieser Service als SentencesService bezeichnet.

35 Details zu JAX-WS 2.2, Siehe Kapitel 3.3. Es wird hier nicht näher darauf eingegangen, da lediglich der Vergleich zwischen SOAP und REST im Vordergrund steht.

36 JAX-RS ist eine Programmierschnittstelle für Java, die es ermöglicht den im vorangegangenen Abschnitt erlauterten, REST Architekturstil zu verwenden. Die Grundlagen für die Entwicklung RESTful Services wurden aus (Burke, 2010) entnommen.

³⁷Zum Zeitpunkt der Durchführung der Tests, beinhaltete die Datenbank Datensätze der Bibel in Altgriechischer Sprache.
### KAPITEL 2. WEB SERVICE TECHNIKEN

<table>
<thead>
<tr>
<th>Technologieart</th>
<th>SOAP</th>
<th>REST</th>
</tr>
</thead>
<tbody>
<tr>
<td>Protokollframework</td>
<td>Protokollframework</td>
<td>Architekturvorschlag</td>
</tr>
</tbody>
</table>

<table>
<thead>
<tr>
<th>Sichtweise des Web</th>
<th>Möglichkeit zum Transport von Nachrichten</th>
<th>Möglichkeit zum Zugriff auf Informationen</th>
</tr>
</thead>
<tbody>
<tr>
<td>Fokus</td>
<td>auf Design von integrierten verteilten Systemen</td>
<td>auf Performance und Skalierbarkeit von verteilten Hypermedia-Systemen</td>
</tr>
</tbody>
</table>

<table>
<thead>
<tr>
<th>Kommunikation</th>
<th>indirekt (über SOAP Intermediär), direkt (Client-Server)</th>
<th>direkt (Client-Server)</th>
</tr>
</thead>
</table>

<table>
<thead>
<tr>
<th>Transport</th>
<th>Protokoll unabhängig</th>
<th>HTTP bzw. HTTPS</th>
</tr>
</thead>
</table>

<table>
<thead>
<tr>
<th>Session Management</th>
<th>Server kann Konversationen über mehrere Request-Response-Zyklen aufrecht erhalten</th>
<th>Server ist zustandslos</th>
</tr>
</thead>
</table>

| Design | 1. Serviceoperationen identifizieren  
2. Datenmodell für den Inhalt der Nachrichten definieren  
3. Transportprotokoll wählen  
4. Implementierung und deployen des Web Services auf Web Service Container | 1. Ressourcen identifizieren  
2. URLs definieren  
3. HTTP-Methoden für Ressource definieren  
4. Beziehungen zwischen einzelnen Ressourcen identifizieren  
5. Implementierung und deployen auf Web Server |
|--------|-------------------------------------------------|------------------------|

<table>
<thead>
<tr>
<th>Naming</th>
<th>kein standardisierter Namingmechanismus vorhanden</th>
<th>konsistenter Namingmechanismus für Ressourcen</th>
</tr>
</thead>
</table>

<table>
<thead>
<tr>
<th>Schnittstellen</th>
<th>Jede Operation bzw. Funktion muss eigenen Service darstellen</th>
<th>global (über URIs)</th>
</tr>
</thead>
</table>

<table>
<thead>
<tr>
<th>Schnittstellenbeschreibung</th>
<th>über WSDL</th>
<th>WADL/WSDL</th>
</tr>
</thead>
</table>

<table>
<thead>
<tr>
<th>Verzeichnisdienst</th>
<th>UDDI</th>
<th>Suchmaschine (z. B.: <a href="http://ecosia.org/">http://ecosia.org/</a>)</th>
</tr>
</thead>
</table>

<table>
<thead>
<tr>
<th>zuverlässige Nachrichtenübertragung</th>
<th>WS-ReliableMessaging, WS-Reliability</th>
<th>POST Exactly Once, ansonsten nicht weiter nötig, da PUT, DELETE, GET idempotent</th>
</tr>
</thead>
</table>

<table>
<thead>
<tr>
<th>Transaktionen</th>
<th>WS-Transaction</th>
<th>über Transaktionsmanager</th>
</tr>
</thead>
</table>

Tabelle 2.7: Zusammenfassung der Unterschiede zwischen SOAP und REST

Der Client ruft den SOAP- bzw. RESTful-Service über eine Uniform Resource Locator (URL), wie in dem Beispiel 2.7 bzw. in 2.8 gezeigt ist, auf.

```plaintext
http://aspra26.informatik.uni-leipzig.de:8080/
SoapRestComparator/SentencesService

Listing 2.7: Aufruf des SOAP-Services
```

```
http://aspra26.informatik.uni-leipzig.de:8080/
```
Bei Aufruf des SOAP-Service schickt der Client dem Server ein SOAP-Datei, die die gewünschte Start- und End-Id beinhaltet. Die Angabe der beiden Ids erfolgt beim RESTful-Service direkt in der URL.

Der Client wurde mit JMeter erzeugt. Dieser generierte pro Nachrichtengröße, 1.000 Requests. Die Größe der Nachricht wurde dabei nicht direkt an der Größe in MB festgelegt, sondern anhand der Anzahl der Sätze, die angefragt wurden und sich schließlich im Antwortdokument befanden. Dabei existierten mit 1-, 100-, 1.000-, 10.000- und 100.000 Sätzen im Antwortdokument fünf verschiedene Nachrichtengrößen. Es wurde immer nur ein Request an den Server gestellt. Der Client erzeugte erst eine neue Anfrage, sobald er eine Antwort empfangen hat.

Zur Verifizierung der Testergebnisse wurden die Tests mehrfach und auf verschiedene Art und Weise durchgeführt. Dies wurde zu Beginn der Arbeit erwähnt.

2.4.2 Versuchsergebnisse


<table>
<thead>
<tr>
<th>Anzahl der Elemente</th>
<th>Gesamtverarbeitungszeit (in ms)</th>
<th>Serververarbeitungszeit (in ms)</th>
<th>Durchsatz (in Requests/Sekunde)</th>
<th>Nachrichtengröße (in KB)</th>
</tr>
</thead>
<tbody>
<tr>
<td>1</td>
<td>8</td>
<td>4</td>
<td>85,9</td>
<td>0,8</td>
</tr>
<tr>
<td>100</td>
<td>16</td>
<td>8</td>
<td>51,5</td>
<td>29,8</td>
</tr>
<tr>
<td>1.000</td>
<td>38</td>
<td>27</td>
<td>24,5</td>
<td>269</td>
</tr>
<tr>
<td>10.000</td>
<td>467</td>
<td>264</td>
<td>2,1</td>
<td>3.347</td>
</tr>
<tr>
<td>100.000</td>
<td>7.364</td>
<td>2.186</td>
<td>0,1</td>
<td>27.961</td>
</tr>
</tbody>
</table>

Tabelle 2.8: Versuchsergebnisse von SOAP

Aus diesen Testresultaten wird deutlich, dass die Dauer eines Request-Response-Zyklus bei SOAP bis zu 60% größer ist als bei REST. Dies ist vor allem dem Overhead an Daten zu zuschreiben, die im SOAP Request- und Responsedokument enthalten sind. Denn das Respondedokument ist bis zu 18% größer als das von REST. Das Requestdokument ist bei SOAP zwar lediglich nur maximal 303 Bytes groß, jedoch existiert für REST kein solches Dokument. Beide dieser Tatsachen haben zur Folge, dass sich die Netzwerklasst bei SOAP erhöht und mehr Daten übertragen werden müssen. Dies wiederum erhöht die Dauer der Request-Response-Zyklen.
KAPITEL 2. WEB SERVICE TECHNIKEN

<table>
<thead>
<tr>
<th>Anzahl der Elemente</th>
<th>Gesamtverarbeitungszeit (in ms)</th>
<th>Serververarbeitungszeit (in ms)</th>
<th>Durchsatz (in Requests/Sekunde)</th>
<th>Nachrichtengröße (in KB)</th>
</tr>
</thead>
<tbody>
<tr>
<td>1</td>
<td>5</td>
<td>3</td>
<td>177,0</td>
<td>0,4</td>
</tr>
<tr>
<td>100</td>
<td>7</td>
<td>6</td>
<td>120,8</td>
<td>24,2</td>
</tr>
<tr>
<td>1.000</td>
<td>28</td>
<td>25</td>
<td>34,2</td>
<td>217</td>
</tr>
<tr>
<td>10.000</td>
<td>240</td>
<td>228</td>
<td>4,2</td>
<td>2.863</td>
</tr>
<tr>
<td>100.000</td>
<td>2.203</td>
<td>1.985</td>
<td>0,5</td>
<td>23.129</td>
</tr>
</tbody>
</table>

Tabelle 2.9: Versuchsergebnisse von REST


In Abbildung 2.4 wurden die durchschnittlichen Gesamtverarbeitungszeiten normalisiert und grafisch dargestellt.

Abbildung 2.4: Durchschnittliche Gesamtverarbeitungszeit pro Element

Aus dieser Grafik geht hervor, dass ab einer Größe von 1.000 Elementen in einem Dokument die Verarbeitungszeiten der REST-basierten Anfragen mit ca. 0,02 Millisekunden relativ konstant sind. Bei SOAP-basierten Anfragen hingegen steigen die Verarbeitungszeiten ab 1.000 Elementen in einem Dokument auf ca. 0,08 Millisekunden pro Element an. Dies bestätigt erneut die Behauptungen, dass die Anfragen in REST schneller verarbeitet werden als in SOAP.
2.5 Zusammenfassung

Die Aufgabe in diesem Kapitel war es, SOAP und REST miteinander in Relation zu setzen. Dabei wurde neben den Grundlagen und Features der beiden Techniken auch die Performance getestet.


Wenn es lediglich um die Verwendung einfacher Datenbank-Lookups oder Datenmanipulationen geht, ist REST, aufgrund der Resultate, die bessere Alternative. REST verwendet ausschließlich ausgereifte und weit verbreitete Standards, wie zum Beispiel HTTP und URIs. Außerdem sind HTTP-Bibliotheken in verschiedenen Programmiersprachen eher zu finden als SOAP Frameworks. Ein weiterer Point, der für REST spricht, ist die geringere Lernkurve. SOAP sollte dann eingesetzt werden, wenn der Service komplex wird, was zum Beispiel bei langandauernden Transaktionen der Fall ist. Obwohl SOAP sehr komplex und mächtig ist, wird es von einer Vielzahl von Tools unterstützt, was wiederum für SOAP spricht. Jedoch benötigt man für REST keine weiteren Tools, als eine HTTP-Bibliothek für die verwendete Programmiersprache. Die Vorteile von SOAP und REST verdeutlichen, dass die Schwierigkeit darin besteht, sich auf eine Technologie festzulegen. Anstatt sich somit für eine der beiden Technologien zu entscheiden, ist das zur Verfügung stellen beider Technologien wohl die beste Lösung.

Zu Beginn dieses Kapitels werden die oben genannten Frameworks auf ihre Architektur und Features untersucht. Im Anschluss daran werden sie in Relation zueinander gesetzt. Den Abschluss bildet dann die Analyse der Performance.

### 3.1 Apache Axis 1


#### 3.1.1 Architektur


---

¹Siehe (Foundation, 2005).


Auf der Seite des Clients ist der Nachrichtenfluss ähnlich wie auf der Serverseite. Wie in Abbildung 3.2 verdeutlicht wird, ist lediglich die Reihenfolge der Chains umgekehrt. Auf der Clientseite wird der Message-Context im Request Flow durch die Anwendung an die Engine übergeben. Im Gegensatz zur Serverseite ist hier die erste Anlaufstation die servicespezifische Kette, die jedoch keinen Provider beinhaltet. Danach werden die Handler der globalen-
Abb. 3.2: Verarbeitung von SOAP-Nachrichten in Apache Axis 1 auf der Clientseite


**Subsysteme**

Um Apache Axis 1 modular aufzubauen, wurden verschiedene Subsysteme eingeführt, die nach ihren Zuständigkeiten unterteilt wurden und zusammenarbeiten können. Jedes dieser Systeme ist unabhängig voneinander, sie können dennoch gemeinsam verwendet werden. Diese Modularität wurde allerdings bei der Entwicklung nicht konsequent durchgeführt. So ist im Quellcode ersichtlich, dass einige Systeme über verschiedene Pakete verteilt sind und einige in anderen Systemen mehrfach verwendet werden. In den nachfolgenden Abschnitten werden einige der wichtigsten Subsysteme vorgestellt. ⁴

**Message Flow Subsystem**


⁴Vgl. (Project, 2005).
Message Model Subsystem

In Apache Axis 1 existieren Klassen, die den Elementen einer SOAP-Nachricht entsprechen. Somit gibt es zum Beispiel eine Klasse `SOAPHeaderElement`, die die Attribute `actor` und `mustUnderstand` beinhaltet oder eine Klasse `SOAPBody`, die die Nutzdaten einer SOAP-Nachricht enthalten.

Während der Deserialisierung wird ein Baum aus SOAP-Klassen erstellt. Dieser wird in Abbildung 3.3 präsentiert.

![Abbildung 3.3: SOAP Elemente in Apache Axis 1](image)

Die Klasse, die hauptsächlich für das Parsen von XML verantwortlich ist, ist `DeserializationContext`. Sie koordiniert den Erstellungsprozess des Baumes und enthält unter anderem einen Stack von SAX-Handlern und eine Referenz auf die Klasse, die die Ergebnisse der Nachrichtenverarbeitung enthält.

### 3.1.2 Data Binding

Apache Axis 1 liefert ein eigenes Type-Mapping-Framework, mit dem die Umwandlung der Datentypen der XML-basierten SOAP-Nachrichten in Java-Typen und umgekehrt erfolgt, welches in diesem Abschnitt erläutert wird.


---

6 Vgl. (Project, 2005).
3.1.3 Features und Erweiterungen von Apache Axis 1

Realisierung von Web Service Erweiterungen

Um WS-* Spezifikationen einsetzen zu können, müssen Handler, die sich um die Sicherheit, Integrität und Vertraulichkeit der Web Services kümmern, von Hand implementiert werden. Jedoch werden einige Frameworks von Apache Web Services Project angeboten, die diese Arbeit abnehmen, elche nachfolgend kurz aufgelistet und erläutert werden.

WS-Coordination, WS-AtomicTransaction und WS-BusinessActivity:

- Wird durch Kandula\(^9\) implementiert.
- Bietet Interoperabilität mit anderen WS-Coordination, WS-AtomicTransaction und WS-BusinessActivity Implementierungen, speziell derer von .NET und IBM.

WS-ReliableMessaging:

- Wird durch Sandesha\(^10\) implementiert.
- Veröffentlicht durch IBM, Microsoft, BEA und TIBCO.
- Unterstützt WS-Policy und WS-Addressing.
- Interoperabilität mit anderen WS-ReliableMessaging Implementierungen.
- Stellt inOrder-Nachrichtenübermittlung zur Verfügung.

WS-Security:

- Wird durch WSS4J implementiert.\(^11\)
- Ist eine Implementierung der OASIS WS-Security.

Nachrichtenoptimierung

Für die Realisierung von SOAP with Attachments müssen in Apache Axis 1 lediglich mit mail.jar und activation.jar zwei Bibliotheken in die Webanwendung aufgenommen werden. Um das Attachment einer SOAP-Nachricht auszulesen bzw. in eine Nachricht zu verpacken, existieren die beiden Klassen JAFDataHandlerSerializerFactory und JAFDataHandlerDeserializerFactory. Wenn ein Service Attachments erzeugt, erstellt der Serializer eine externe Datei im DIME bzw. MIME Format. Über die Klasse DataHandler wird schließlich die Attachmenterzeugung bzw. der Empfang gesteuert.\(^12\)

\(^12\)Vgl. (Dapeng Wang, 2004) Seite 473 - 481
Deployment


Die Basis für das benutzerdefinierte Deployment bildet die kompilierte Version des Services und ein Web Service Deployment Descriptor (*WSDD*). Ein WSDD bündelt die Methoden, die es in Axis zu veröffentlichen gilt. Im folgenden Listing 3.1 ist ein Beispiel einer WSDD-Datei dargestellt.

```
<ns1:deployment xmlns="http://xml.apache.org/axis/wsdd/
    xmlns:java="http://xml.apache.org/axis/wsdd/providers/java"
    xmlns:ns1="http://xml.apache.org/axis/wsdd/">
  <ns1:globalConfiguration>
    <ns1:parameter name="disablePrettyXML" value="true"/>
    <ns1:parameter name="adminPassword" value="admin"/>
    ...<ns1:requestFlow> ...</ns1:requestFlow>
  </ns1:globalConfiguration>
  ...
  <ns1:service name="Sentences" provider="java:RPC"
    style="wrapped" use="literal">
    <ns1:operation name="getSentences" qname="ns3:getSentences"
      returnQName="ns3:getSentencesReturn"
      returnType="xsd:string" soapAction=""
      xmlns:ns3="http://ServiceTypes"
      xmlns:xsd="http://www.w3.org/2001/XMLSchema">
      <ns1:parameter qname="ns3:scriptId" type="xsd:int"/>
      <ns1:parameter qname="ns3:endId" type="xsd:int"/>
    </ns1:operation>
    <ns1:parameter name="allowedMethods" value="getSentences"/>
    ...<ns1:parameter name="className"
      value="ServiceTypes.Sentences"/>
  </ns1:service>
  <ns1:transport name="http"/>
  ...
</ns1:deployment>
```

Listing 3.1: Ausschnitt eines Web Service Deployment Descriptor

### 3.2 Apache Axis 2

Apache Axis 2 ist der Nachfolger von Apache Axis 1. Es unterstützt sowohl SOAP als auch RESTful Services. Apache Axis 2 wurde von Grund auf neu entwickelt. Die Vorteile, die daraus resultieren, sind unter anderem eine höhere Performance, einfachere Erweiterbarkeit und leichtere Konfiguration. Des Weiteren unterstützt Axis 2 HTTP, TCP, JMS und SMTP als Transportprotokolle.\(^{16}\)

#### 3.2.1 Architektur von Apache Axis 2


Die Architektur von Apache Axis 2 ist modular aufgebaut und unterteilt sich in Kernmodule und Zusatzmodule. Die Kernmodule bilden die Kernarchitektur und die Grundlage für die Zusatzmodule. Die folgenden Tabellen 3.1 und 3.2 zeigen einen Überblick über diese.


---

\(^{14}\) Vgl. (Foundation, 2005).

\(^{15}\) Es existieren noch zahlreiche weitere Möglichkeiten, Services über WSDD zu konfigurieren. Da diese alle zu nennen nicht Teil dieser Arbeit ist, sei an dieser Stelle auf (Project, 2006).

\(^{16}\) Vgl. (Thilo Frotscher, 2007) S. 21 und (Grimm, 2007).

\(^{17}\) Siehe Kapitel 3.2.1.
### Module Beschreibung

<table>
<thead>
<tr>
<th>Module</th>
<th>Beschreibung</th>
</tr>
</thead>
<tbody>
<tr>
<td>XML Processing Model</td>
<td>Es handelt sich hierbei um die wichtigste und komplexeste Aufgabe: die Verarbeitung der SOAP-Nachrichten. Es wurde in ein separates Projekt ausgelagert, namens AXIOM.</td>
</tr>
<tr>
<td>SOAP Processing Model</td>
<td>Diese Komponente kontrolliert die Ausführung der Verarbeitung. Es werden hier verschiedene Phasen der Ausführung definiert, die durchlaufen werden müssen. Der Benutzer hat hier die Möglichkeit das Modul zu erweitern.</td>
</tr>
<tr>
<td>Deployment Model</td>
<td>Es ermöglicht es dem Benutzer Services zu verteilen, die Beförderung der Nachrichten zu konfigurieren und das SOAP Processing Model zu erweitern.</td>
</tr>
<tr>
<td>Client API</td>
<td>Dies ist die API, die es ermöglicht mit Web Services über Apache Axis 2 zu kommunizieren.</td>
</tr>
<tr>
<td>Transports</td>
<td>In Apache Axis 2 existiert ein Transport Framework was dem Benutzer ermöglicht, verschiedene Transportmöglichkeiten zu konfigurieren. Das Framework fügt sich in das SOAP Processing Model ein.</td>
</tr>
</tbody>
</table>

Tabelle 3.1: Kernmodule von Apache Axis 2

<table>
<thead>
<tr>
<th>Module</th>
<th>Beschreibung</th>
</tr>
</thead>
<tbody>
<tr>
<td>Code Generation</td>
<td>Dies ist ein Tool, womit es möglich ist, clientseitigen- sowie serverseitigen Code generieren zu lassen.</td>
</tr>
<tr>
<td>Data Binding</td>
<td>Serialisierung und Deserialisierung</td>
</tr>
</tbody>
</table>

Tabelle 3.2: Zusatzmodule von Apache Axis 2

Descriptionhierarchie zu sehen. In der anschließenden Tabelle 3.3 werden die Hierarchien, des Information Models kurz erläutert. Dabei ist anzumerken, dass eine untergeordnete Konfiguration eine übergeordnete Konfiguration überschreiben kann.
Abbildung 3.4: Beziehung zwischen Context- und Description-Hierarchie

<table>
<thead>
<tr>
<th>Kontext</th>
<th>Beschreibung</th>
<th>Konfiguration</th>
<th>Beschreibung</th>
</tr>
</thead>
<tbody>
<tr>
<td>Configuration-Context</td>
<td>Ist die Wurzel der Kontexthierarchien und ein Container für globale Laufzeitinformationen</td>
<td>Axis-Configuration</td>
<td>Ist die Wurzel der Description-Hierarchie und dient als Container für Konfigurationsinformationen.</td>
</tr>
<tr>
<td>ServiceGroup-Context</td>
<td>Beinhaltet Informationen über die jeweiligen Servicegruppen</td>
<td>AxisService-Group</td>
<td>Diese Instanz repräsentiert eine Servicegruppe</td>
</tr>
<tr>
<td>Service-Context</td>
<td>Beinhaltet den Kontext eines Services und ist in dem jeweiligen Service verwendbar</td>
<td>AxisService</td>
<td>Beinhaltet Operationen und die Konfiguration des Servicelevels</td>
</tr>
<tr>
<td>Operation-Context</td>
<td>Beinhaltet Information über das Nachrichtenaustauschformat</td>
<td>AxisOperation</td>
<td>Beinhaltet Konfiguration des Operationlevels</td>
</tr>
<tr>
<td>Message-Context</td>
<td>Beinhaltet alle Informationen über die derzeitig ausgeführte Nachricht</td>
<td>AxisMessage</td>
<td>Beinhaltet statische Informationen, wie das Schema der Nachricht</td>
</tr>
</tbody>
</table>

Tabelle 3.3: Information Model Hierarchien in Apache Axis 2

In den nachfolgenden Abschnitten werden einige der bereits angesprochenen Module genauer betrachtet.
XML Processing Model


AXIOM vereint die Vorteile von ereignisbasierten- und baumbasierten APIs. Ursprünglich war es als ein Mechanismus zur Sammlung und Zwischenspeicherung für StAX-Ereignisse vorgesehen. Mit Hilfe dieses Mechanismus war es möglich, dass diese Ereignisse später für die Verarbeitung wieder herangezogen werden können. Aufgrund der hohen Flexibilität, die dadurch geboten wird, wurde es zu einem kompletteten Objektmodell für XML-Infosets aufgebaut.


Die Architektur von AXIOM stellt sich wie folgt dar: Es existiert ein Interface (Builder), dass für den Aufbau des Objektmodells zuständig ist. Für dieses Interface werden mehrere Implementierungen geliefert, die neben der Erstellung von spezifischen SOAP Nachrichten und der Generierung von Attachments, auch Ereignisse von SAX verarbeiten können. Für die Verwaltung der Objektmodelle wurden während der Entwicklungszzeit mehrere Speichermode
d ausgetestet. Es wurde sich schließlich für eine Variante entschieden, die mit verketteten Listen arbeitet. Jedoch hat der Entwickler die Möglichkeit sein eigenes Speichermodell zu integrieren.\(^{18}\)

SOAP Processing Model


Es bleibt anzumerken, dass für jede SOAP Nachricht diese Abarbeitungsreihenfolge vollzogen wird.


19Es können auch die Klassen ServiceClient oder OperationClient der Client-API verwendet werden.
Abbildung 3.5: Verarbeitung von SOAP-Nachrichten in Apache Axis 2

Handler des OutFlows.
Eingehende und ausgehende Nachrichten werden der Reihe nach durch die in den Tabellen 3.4 und 3.5 ersichtlichen Phasen geleitet.

<table>
<thead>
<tr>
<th>Phasen</th>
<th>Beschreibung</th>
</tr>
</thead>
<tbody>
<tr>
<td>2. Pre-Dispatch Phase</td>
<td>Befüllen des MessageContext, um Dispatching einzuleiten.</td>
</tr>
<tr>
<td>3. Dispatch Phase</td>
<td>Sucht den korrekten Service und die dazugehörige Operation.</td>
</tr>
<tr>
<td>4. benutzerdefinierte Phase</td>
<td>Hier werden die vom Entwickler hinzugefügten Handler durchlaufen.</td>
</tr>
</tbody>
</table>

Tabelle 3.4: Phasen einer eingehenden Nachricht in Apache Axis 2

Das SOAP Processing Model in Apache Axis 2 kann durch das Hinzufügen von Handlern und Modulen erweitert werden.\textsuperscript{20}

\subsection{3.2.2 Data Binding}

Ein großer Vorteil von Apache Axis 2 ist die Erweiterbarkeit des Data Bindings. Es befindet sich nicht im Kern von Apache Axis 2. Somit ist es möglich, mit Hilfe des \textit{Code Generator}

\textsuperscript{20}Vgl. (Thilo Frotscher, 2007) Kapitel 9 Seite 245 - 263.
### KAPITEL 3. WEB SERVICE FRAMEWORKS

<table>
<thead>
<tr>
<th>Phasen</th>
<th>Beschreibung</th>
</tr>
</thead>
<tbody>
<tr>
<td>2. Benutzerphase</td>
<td>Führt Handler der benutzerdefinierten Phase aus.</td>
</tr>
<tr>
<td>3. Transportphase</td>
<td>Senden der Nachricht zum Ziel.</td>
</tr>
</tbody>
</table>

<table>
<thead>
<tr>
<th>Frameworks</th>
<th>Beschreibung</th>
</tr>
</thead>
<tbody>
<tr>
<td>ADB</td>
<td>Leichtgewichtiger Schema Compiler, der mit StAX arbeitet.</td>
</tr>
<tr>
<td>JAXB-RI</td>
<td>Ist die Referenzimplementierung der einzigen Spezifikation für XML Data Binding in Java.</td>
</tr>
<tr>
<td>JibX</td>
<td>Zeichnet sich durch seine hohe Performanz aus.</td>
</tr>
</tbody>
</table>

Tabelle 3.5: Phasen einer ausgehenden Nachricht in Apache Axis 2

Frameworks andere Data Binding Frameworks problemlos einzubinden. In der Tabelle 3.6 sind die zur Verfügung stehenden Frameworks in Apache Axis 2 angegeben.

Framework Entwicklung von RESTful Services


```xml
...  
<parameter name="servicePath">services</parameter>
<parameter name="restPath">rest</parameter>
```

KAPITEL 3. WEB SERVICE FRAMEWORKS

Listing 3.2: Ausschnitt axis2.xml für REST Support

In dieser Einstellung wird der REST-Support in Apache Axis 2 freigeschalten, was jedoch bereits in der Standardeinstellung so konfiguriert ist. Dabei besagt der Wert `disableREST`, wenn er auf true gesetzt wird, dass REST-Support für beide Endpunkte deaktiviert ist. `enableRESTInAxis2MainServlet` besagt, wenn der Wert auf true steht, dass das AxisServlet unter `/services/*` auch REST-basierte Anfragen entgegen nimmt. `disableSeparateEndpointForREST` aktiviert bzw. deaktiviert einen separaten Endpunkt für REST-basierte Anfragen.22

Realisierung von Web Service Erweiterungen


**WS-Coordination, WS-AtomicTransaction und WS-BusinessActivity:**

- Wird durch Kandula23 implementiert.
- Bietet Interoperabilität mit anderen WS-Coordination, WS-AtomicTransaction und WS-BusinessActivity Implementierungen, speziell derer von .NET und IBM.

**WS-ReliableMessaging:**

- Wird durch Sandesha24 implementiert.
- Veröffentlicht durch IBM, Microsoft, BEA und TIBCO.
- Wurde für Apache Axis 2 entwickelt.
- Bietet Interoperabilität mit anderen WS-ReliableMessaging Implementierungen.

**WS-ResourceFrameworks, WS-BasNotification und WS-DistributedManagement:**

- Wird durch Muse25 implementiert.

---

KAPITEL 3. WEB SERVICE FRAMEWORKS

- Ist konform mit WS-Addressing und SOAP 1.2.

**WS-Security:**
- Wird durch Rampart implementiert.
- Ist eine Erweiterung zu WSS4J.
- Es wird zusätzlich das Paket Apache XML Security benötigt.

**WS-Policy:**
- Wird durch Neethi implementiert.
- Ist auch integriert im Apache WS-Commons Projekt.
- Auf der Clientseite werden, wenn die WSDL Policies enthält, im Code entsprechende Erweiterungen hinzugefügt.
- Auf der Serverseite wird das WSDL-Dokument um alle in der axis.xml-Datei befindlichen Policies erweitert.

**addressing:**
- Implementiert WS-Addressing.
- Ist bereits Standardmäßig im modules-Ordner von Axis 2 enthalten und ist mit allen anderen Web Services verfügbar.
- Auf Clientseite muss lediglich ein Repository erstellt werden, in dem das Modul hinzugefügt wird.

**Nachrichtenoptimierung**
Für die Nachrichtenoptimierung ist AXIOM in Apache Axis 2 verantwortlich. Dafür gibt es die Klasse OMText, die sowohl Text als String als auch Binärdaten entgegennehmen kann. Um ein OMText-Objekt für Attachment zu erzeugen, muss entweder ein Objekt vom Typ DataHandler oder eine Zeichenkette vom Typ String, der ein in Base64-kodiertes Attachment darstellt, übergeben werden.  

**Deployment**


3.3 Metro 2.0


3.3.1 Architektur

In Abbildung 3.6 ist erkennbar, dass Metro 2.0 aus drei Ebenen, die sich auf zwei Komponenten verteilen, besteht. Die untersten beiden Ebenen Web Service Core und XML Processing werden durch die Referenzimplementierung von JAX-WS realisiert. Die obere durch WSIT. Durch WSIT ist es u.a. in Metro möglich, sichere Nachrichtenübertragung oder Verschlüsselung der Nachrichten zu implementieren.

Abbildung 3.6: Metro Web Services Stack


Um SOAP-Nachrichten zu erzeugen, benutzt JAX-WS die SOAP with Attachments API for Java (SAAJ). Um die Umwandlung von XML-Datentypen in Java-Datentypen vorzunehmen, delegiert JAX-WS an JAXB weiter. Web Services werden bei JAX-WS mit Hilfe von Annotationen entwickelt. Jeder Service muss dabei entweder mit @WebService oder mit @WebServiceProvider annotiert sein (siehe Listing 3.3).

```java
@WebService (name = "SentencesService")
public class Sentences {
    @WebMethod()
    public String[] getSentences (
        @WebParam(name = "startId") int startId ,
        @WebParam(name = "endId") int endId) {
        String[] result = null;
        try {
            result = Finder.getSentencesByStartAndEndId (startId , endId);
        } catch (SQLException e) {
            throw new ServiceException("error while database request", e);
        }
        return result;
    }
}
```

Listing 3.3: Implementierung SOAP Services in Metro 2.0

KAPITEL 3. WEB SERVICE FRAMEWORKS

<table>
<thead>
<tr>
<th>Modul</th>
<th>Beschreibung</th>
</tr>
</thead>
<tbody>
<tr>
<td>runtime</td>
<td>Enthält den Web Service Kern.</td>
</tr>
<tr>
<td>tools</td>
<td>Tools, um zum Beispiel aus WSDL Dateien Services zu erzeugen.</td>
</tr>
<tr>
<td>APT</td>
<td>Tool, um Annotationen zu verarbeiten.</td>
</tr>
<tr>
<td>Annotation Processor</td>
<td>Aus APT um Java Quellcode Dateien wie Web Services zu veröffentlichen.</td>
</tr>
<tr>
<td>JAXB XJC-API</td>
<td>Schema Compiler.</td>
</tr>
<tr>
<td>JAXB runtime-API</td>
<td>Teil von JAXB, der den Vertrag zwischen JAXB und JAX-WS definiert.</td>
</tr>
</tbody>
</table>

Tabelle 3.7: Hauptmodule von JAX-WS


Das Data Binding wird an JAXB weitergeleitet. Der Vorgang des Data Bindings wird im folgenden Kapitel beschrieben.\(^{32}\)

3.3.2 Data Binding

Das Data Binding erfolgt in Metro 2.0 über JAXB. Es ist die einzige Spezifikation für XML-Data Binding in Java. Mittels JAXB ist es möglich, ein XML-Schema einzulesen und daraus

automatisch Java-Klassen zu generieren. JAXB besteht wiederum aus mehreren Komponenten, die in Tabelle 3.8 aufgezeigt werden.

<table>
<thead>
<tr>
<th>Komponente</th>
<th>Beschreibung</th>
</tr>
</thead>
<tbody>
<tr>
<td>Binding Compiler</td>
<td>Kern der JAXB Verarbeitung, der ein XML Schema transformiert oder bindet.</td>
</tr>
<tr>
<td>JAXB Binding Framework</td>
<td>Enthält Schnittstellen für Serialisierung, Deserialisierung und Validierung von XML.</td>
</tr>
<tr>
<td>Schema Klassen</td>
<td>Klassen, die durch den Compiler erzeugt wurden.</td>
</tr>
</tbody>
</table>

Tabelle 3.8: Hauptmodule von JAXB


Es bleibt anzumerken, dass JAXB zwei Möglichkeiten der Validierung vorsieht: Die Unmarshal-Time Validierung ermöglicht es, Informationen, wie Fehler und Warnhinweise, die während der Deserialisierung auftraten, zu erhalten. Die zweite Möglichkeit ist die On-Demand Validierung. Sie liefert Fehler und Warnungen, die direkt im Contentbaum vorkamen. Während die Unmarshal-Time Validierung nur zu Beginn der Deserialisierung ausgeführt werden kann, kann die On-Demand Validierung immer angestoßen werden. Dies ist vor allem dann sinnvoll, wenn der Contentbaum modifiziert wurde.33

### 3.3.3 Features von Metro 2.0

#### Entwicklung von RESTful Services

Metro 2.0 bietet ebenfalls Unterstützung von RESTful Services, bei der wie bei der Entwicklung von SOAP Services auch auf die Spezifikation JAX-WS zurückgegriffen wird.34

```java
@WebServiceProvider
```

Das Codebeispiel (Listing 3.4) zeigt, dass die Ressource, die es zu veröffentlichen gilt, das Interface Provider implementieren muss. Die Provider Schnittstelle bietet eine einzige Methode an, die es zu überschreiben gilt:

```java
public Source invoke(
    @WebParam(name = "request") Source request) {
    try {
        MessageContext mc =
            m_WsContext.getMessageContext();
        String path =
            (String) mc.get(MessageContext.PATH_INFO);
        String method =
            (String) mc.get(MessageContext.HTTP_REQUEST_METHOD);
        String queryString =
            (String) mc.get(MessageContext.QUERY_STRING);

        String[] keyValuePairs = queryString.split("&");
        int startId = keyValuePairs[0];
        int endId = params.get("endId");

        if (method.equals("GET"))
            return doGet(startId, endId);

        if (method.equals("POST"))
            return doPost(startId, endId, mc);

        if (method.equals("PUT"))
            return doPut(request, mc);

        if (method.equals("DELETE"))
            return doDelete(request, mc);
    }
}
```

Listing 3.4: Implementierung RESTful Services in Metro 2.0

Die @Resource Annotation wird verwendet, damit JAX-WS den WebServiceContext zu der Ressource hinzufügen kann. @BindingType(value=HTTPBinding.HTTP_BINDING) wird benutzt, um zu zeigen, dass die Ressource Sentences mittels HTTP-Binding veröffentlicht wird. Dies ist das Gegenstück zu SOAPBinding.SOAP11HTTP_BINDING oder SOAPBin-
KAPITEL 3. WEB SERVICE FRAMEWORKS

**ding.SOAP12HTTP_BINDING.**
Im letzten Schritt muss nun in der Konfigurationsdatei *sun-jaxws.xml* der Endpunkt bekanntgegeben werden. 35

**Realisierung der Web Service Erweiterungen**
WSIT ist eine Ansammlung von implementierten Technologien, wie zum Beispiel sichere Nachrichtenübertragung und Bootstrapping. Es wurde anfänglich unter dem Namen *Projekt Tango* geführt und ist bis heute unter diesem Namen weit verbreitet. WSIT gilt als die Erweiterung zu JAX-WS und implementiert die in Abbildung 3.7 aufgezeigten Techniken.

![Abbildung 3.7: WSIT Web Service Features](image)


```
wsit -*.xml
```

*Listing 3.5: Namenskonvention der Konfigurationsdatei*

35 Vgl. (Hansen, 2007) S. 85 - 136
Wie die Konfiguration vorzunehmen ist, wurde ausreichend dokumentiert. Des Weiteren bietet ein WSIT-Plugin für NetBeans die Möglichkeit die Konfiguration per Mausklick zu realisieren.\textsuperscript{36} Für Eclipse existieren keine weiteren, nennenswerten Plugins. \textsuperscript{37}

\textbf{Nachrichtenoptimierung}


\textbf{Deployment}


\begin{verbatim}
<?xml version="1.0" encoding="UTF-8"?>
<endpoints version="2.0" xmlns="http://java.sun.com/xml/ns/jax-ri/runtime">
  <endpoint implementation="ServiceTypes.SoapServices.Sentences"
    name="SentencesService"
    url-pattern="/SentencesService"/>
  <endpoint name="SentencesResource"
    implementation="ServiceTypes.RestResources.Sentences"
    url-pattern="/SentencesResource/*"/>
</endpoints>
\end{verbatim}

Listing 3.6: Beispiel der Konfigurationdatei sun-jax-ws.xml


\textsuperscript{36}Dokumentation zur Konfiguration der Sicherheitsfeatures: https://wsit-docs.dev.java.net/releases/1-0-FCS/WSITTutorial.pdf/.

\textsuperscript{37}Vgl. (Sun Microsystems, 2007) und (Gupta, 2007b).

\textsuperscript{38}Eine detaillierter Beschreibung ist in https://metro.dev.java.net/guide/Deploying_Metro_endpoint.html zu finden.
3.4 Performanceevaluation der Web Service Frameworks

In diesem Abschnitt werden die Details der Implementierungen der Services, die Resultate sowie die Auswertungen der Tests dargestellt. Die aus den vorangegangenen Abschnitten erarbeiteten Unterschiede der Web Service Frameworks sind in der nachfolgenden Tabelle 3.9 zusammengefasst. 39

<table>
<thead>
<tr>
<th></th>
<th>Apache Axis 1</th>
<th>Apache Axis 2</th>
<th>Metro 2.0</th>
</tr>
</thead>
<tbody>
<tr>
<td>Data Binding</td>
<td>eigenes Typemapping Framework (Serialisierung, Deserialisierung auf Basis von SAX)</td>
<td>Unterstützung von ADB(^{40}), JiBX(^{41}), JAXB, JAXME, XML Beans(^{42})</td>
<td>JAXB</td>
</tr>
<tr>
<td>Unterstützung von WS-*</td>
<td>Integrierbar durch Module (Kandula, Kandesha, WSS4J)</td>
<td>Integrierbar durch Module (Kandula2, Muse, Sandesha2, Rampart, WS-Commons)</td>
<td>Bereits in WSIT enthalten</td>
</tr>
<tr>
<td>Unterstützung RESTful Services</td>
<td>Nein</td>
<td>Konfigurierbar in axis2.config</td>
<td>Implementierbar durch JAX-WS</td>
</tr>
<tr>
<td>Deployment</td>
<td>Automatisch (.jws), benutzerdefiniert (.war und server-config.wsdd)</td>
<td>Als .Axis Archive und .war, hot deployment und hot update</td>
<td>Als .war und sun-jaxws.xml, hot deployment</td>
</tr>
<tr>
<td>IDE Unterstützung</td>
<td>Mäßig in NetBeans 6.x und Eclipse</td>
<td>Mäßig in NetBeans 6.x und Eclipse</td>
<td>Sehr gut in NetBeans 6.x, mäßig in Eclipse</td>
</tr>
<tr>
<td>Java Unterstützung</td>
<td>Ab Version 1.4</td>
<td>Ab Version 1.4</td>
<td>Ab Version 1.5</td>
</tr>
<tr>
<td>Dokumentation</td>
<td>Sehr gut</td>
<td>Sehr gut</td>
<td>Sehr gut</td>
</tr>
<tr>
<td>Open Source</td>
<td>Ja</td>
<td>Ja</td>
<td>Ja</td>
</tr>
<tr>
<td>Attachments</td>
<td>SwA</td>
<td>SwA und MTOM</td>
<td>SwA und MTOM</td>
</tr>
<tr>
<td>Erweiterbarkeit</td>
<td>Einfach, durch Entwicklung weiterer Handler bzw. Integration von Modulen</td>
<td>Einfach durch Entwicklung weiterer Handler bzw. Integration von Modulen</td>
<td>Schwierig durch Entwicklung weiterer Handler</td>
</tr>
</tbody>
</table>

Tabelle 3.9: Zusammenfassung der Features der Frameworks

Integration der Web Service Frameworks

Bei der Arbeit mit den Frameworks ist erwähnenswert, dass sie alle sehr einfach in den Tomcat Version 6.0.14 zu integrieren waren. Metro 2.0 bot eine build.xml an, die automatisch Metro 2.0 in Tomcat integrierte. Die Datei TOMCAT_HOME/conf/catalina.properties


**SOAP-Dateien**


**WSDL-Dateien**


### 3.4.1 Service-Implementierung


Abbildung 3.8: Klassendiagramm eines Services und einer Ressource

Als Data Binding Komponente wurde für den Kernel auf Basis von Apache Axis 1 das von Axis 1 implementierte Typemapping-Framework, für den Kernel auf Basis von Apache Axis 2 Axis Data Binding (ADB) und für den Kernel auf Basis von Metro 2.0 JAXB, verwendet.

3.4.2 Versuchsergebnisse


Versuchsergebnisse der Loadtests

Im folgenden sind die Ergebnisse der Tests tabellarisch aufgezeigt. Der Client wurde mit JMeter erzeugt. Dieser stellte sequentiell eine neue Abfrage an eine der drei Web-Anwendungen. Dabei variierte die Größe des angefragten Intervalls, zwischen 1-, 100-, 1.000-, 10.000- und 100.000 Elementen, welche jeweils 1.000 Mal ausgeführt wurden. Die Gesamtverarbeitungszeit ist dabei die Zeit, die benötigt wird, einen kompletten Request-Response-Zyklus zu durchlaufen. Sie gibt somit an, wie viel Zeit vom Senden des Request bis zum Empfang des Responses vom Server vergeht. Die Serververarbeitungszeit ist die Zeit, die der Server benötigt, um eine der Anfrage entsprechende Antwort zu generieren. Hier fallen z.B. auch die Zeiten der Datenbankabfragen oder der Berechnung mit hinein. Der Durchsatz hingegen spiegelt die gestellten Anfragen pro Sekunde wider.

Versuchsergebnisse der Stresstests


Getestet wurden bei den Stresstests:

- die Anzahl der gestellten Requests pro Minute,
- die durchschnittliche Verarbeitungszeit eines Requests,
- die aufgetretenen Systemfehler während der Verarbeitung,
- die aktiven Benutzer auf dem Server,
- der Speicherverbrauch der Web-Anwendung und
- die Auslastung der CPU während der Verarbeitung der Requests.

Bei der Auswertung fällt vor allem auf, dass auch hier die Ergebnisse der Services, die mit Apache Axis 1 erzeugt wurden, eine sehr viel schlechtere Performance aufweisen als die von Metro 2.0 und Apache Axis 2.

Die maximale Requestrate bei Apache Axis 1 liegt bei ca. 1.600 Anfragen, bei Apache Axis 2 bei ca. 6.500 und bei Metro 2.0 bei ca. 7.500. Die Requestrate spiegelt dabei die pro Minute verarbeiteten Anfragen des Servers wieder und beinhaltet auch die fehlerhaften Anfragen, die im Diagramm der Systemfehler dargestellt sind.

Die durchschnittliche Verarbeitungszeit ist die Zeit, die der Server vom Empfang des Requests bis zum Verschicken der Antwort benötigt. Aus den Diagrammen geht hervor, dass Apache Axis 1 nach 2.500 Minuten durchschnittlich ca. 4.000 Millisekunden für die Verarbeitung der Anfrage benötigt. Apache Axis 2 braucht dagegen maximal durchschnittlich ca. 1.600 Millisekunden und Metro 2.0 durchschnittlich ca. 1.250 Millisekunden.

Die Messung der Systemfehler ergibt, dass der erste Fehler bei Apache Axis 1 nach 1.700 Minuten auftritt. Bei Apache Axis 2 und Metro 2.0 treten die ersten Fehler schon nach 1.500 Minuten auf. Der Grund dafür liegt in der Anzahl der zu verarbeiteten Requests, denn Apache Axis 2 und Metro 2.0 verarbeiten zu diesem Zeitpunkt schon die vierfache Anzahl an Requests pro Minute.
Tabelle 3.13: Statistiken der Web Service Kernel beim Stresstest (1)
Tabelle 3.14: Statistiken der Web Service Kernel beim Stresstest (2)
Die Gesamtanzahl der Requests geht aus der Tabelle 3.15 hervor:

<table>
<thead>
<tr>
<th></th>
<th>Apache Axis 1</th>
<th>Apache Axis 2</th>
<th>Metro 2.0</th>
</tr>
</thead>
<tbody>
<tr>
<td>Gesamtanzahl der Requests</td>
<td>3,927,466</td>
<td>15,691,734</td>
<td>17,905,524</td>
</tr>
<tr>
<td>durchschnittliche Verarbeitungszeit des Servers (in ms)</td>
<td>2214</td>
<td>850</td>
<td>743</td>
</tr>
<tr>
<td>Gesamtverarbeitungszeit (in ms)</td>
<td>3121</td>
<td>940</td>
<td>812</td>
</tr>
<tr>
<td>Anzahl der Fehler</td>
<td>13,542</td>
<td>6,452</td>
<td>11,064</td>
</tr>
<tr>
<td>durchschnittliche Fehlerquote (in %)</td>
<td>0,4</td>
<td>0,04</td>
<td>0,06</td>
</tr>
</tbody>
</table>

Tabelle 3.15: Zusammenfassung der Rampentestergebnisse

Obwohl Apache Axis 1 weniger Requests pro Minute verarbeitet, liegt die Fehlerquote deutlich höher als bei den andern beiden Web Service Frameworks.

Der maximale Speicher für Tomcat wurde auf 2.048 MB festgelegt. Bei Apache Axis 1 wird der vorher festgelegte Speicher sofort ausgenutzt. Apache Axis 2 und Metro 2.0 sind sich hierbei wieder ähnlich. Der Speicher wird bei diesen Web Services erst nach 1.000 Minuten vollständig ausgelastet.

Eines haben alle drei Web Service Frameworks gemeinsam: die Auslastung der CPU ist bereits nach 10 Nutzern voll erreicht. Der Grund dafür ist, dass die Tests in einer relativ homogenen Umgebung durchgeführt wurden. Server und Client befanden sich im gleichen Netzwerk, was zur Folge hatte, dass die Anfragen und Antworten schneller empfangen und verschickt werden konnten. Somit hatte der Server nicht genügend Zeit, um sich zu erholen.

Vergleich von SOAP und REST innerhalb von Apache Axis 2 und Metro 2.0


<table>
<thead>
<tr>
<th>Anzahl der Elemente</th>
<th>Gesamtverarbeitungszeit (in ms)</th>
<th>Serververarbeitungszeit (in ms)</th>
<th>Durchsatz (in Requests/Sekunde)</th>
</tr>
</thead>
<tbody>
<tr>
<td>1</td>
<td>6 (15)</td>
<td>3 (4)</td>
<td>151,2 (54,5)</td>
</tr>
<tr>
<td>100</td>
<td>8 (15)</td>
<td>6 (7)</td>
<td>111,6 (54,5)</td>
</tr>
<tr>
<td>1.000</td>
<td>25 (36)</td>
<td>24 (26)</td>
<td>38,6 (26,4)</td>
</tr>
<tr>
<td>10.000</td>
<td>228 (274)</td>
<td>(216) (218)</td>
<td>4,4 (3,6)</td>
</tr>
<tr>
<td>100.000</td>
<td>2.150 (6,203)</td>
<td>1.788 (1,646)</td>
<td>0,4 (0,2)</td>
</tr>
</tbody>
</table>

Tabelle 3.16: Versuchsergebnisse der Loadtests von SOAP und REST in Apache Axis 2

Da der Test von SOAP und REST genauso wie die Loadtests erfolgte, wird hier von einer erneuten Erläuterung abgesehen.
<table>
<thead>
<tr>
<th>Anzahl der Elemente</th>
<th>Gesamtverarbeitungszeit (in ms)</th>
<th>Serververarbeitungszeit (in ms)</th>
<th>Durchsatz (in Requests/Sekunde)</th>
</tr>
</thead>
<tbody>
<tr>
<td>1</td>
<td>8 (8)</td>
<td>6 (3)</td>
<td>106,5 (90,6)</td>
</tr>
<tr>
<td>100</td>
<td>11 (17)</td>
<td>10 (6)</td>
<td>83,4 (49,1)</td>
</tr>
<tr>
<td>1.000</td>
<td>33 (33)</td>
<td>30 (24)</td>
<td>29,9 (29,9)</td>
</tr>
<tr>
<td>10.000</td>
<td>269 (292)</td>
<td>263 (214)</td>
<td>3,7 (3,7)</td>
</tr>
<tr>
<td>100.000</td>
<td>2.867 (5.992)</td>
<td>2.628 (1.692)</td>
<td>0,4 (0,2)</td>
</tr>
</tbody>
</table>

Tabelle 3.17: Versuchsergebnisse der Loadtests von SOAP und REST in Metro 2.0


3.5 Zusammenfassung

In diesem Kapitel war die Hauptaufgabe, eine Entscheidungsgrundlage für die Neuentwicklung des bestehenden SOAP-Kernels zu finden. Apache Axis 1 hat wesentlich schlechter abgeschnitten als Apache Axis 2 und Metro 2.0. Jedoch nicht nur bei den Tests der Performance, auch bei den Erweiterungsmöglichkeiten durch bestehende Module ist Apache Axis 1 rückständig.

Aus den oben genannten Gründen ist eine Neuentwicklung des bestehenden SOAP-Kernels auf Basis eines anderen Frameworks anzustreben. Die Entscheidung darüber, ob für die Neuentwicklung Metro 2.0 oder Apache Axis 2 verwendet werden sollte, ist abzuwägen. Zum einen ist Apache Axis 2 ein weit verbreitetes und sehr flexibles Web Service Framework, was jedoch eher schlecht durch Plugins für IDEs unterstützt wird. Zum anderen ist Metro 2.0 ein vergleichsweises starres Framework mit einer sehr guten IDE-Unterstützung in NetBeans 6.x und ebenfalls sehr verbreitet. Hier muss der Entwickler abwägen, ob die Services mit dem .NET Framework kommunizieren sollen oder eine Vielzahl von WS-* Spezifikationen benötigt werden. In diesem Fall sollte eher Metro 2.0 verwendet werden, da es eine bewiesene Interoperabilität zu .NET besitzt, sowie die Module für die WS-* Spezifikation bereits integriert. Wenn dies nicht der Fall ist, so sollte der neue SOAP Kernel eher auf Basis von Apache Axis 2 entwickelt wer-
den, da dies der Nachfolger von Apache Axis 1 ist und aufgrund der Vorerfahrung mit der Entwicklung die Lernkurve wesentlich geringer ausfallen kann.
4 Programmierschnittstellen für XML

4.1 Document Object Model


DOM betrachtet ein XML-Dokument nicht nur als eine Menge von Zeichen, sondern auch als eine geordnete Menge von Objekten. Dafür stellt DOM für alle möglichen Komponenten eines Dokuments eine abstrakte Objektklasse mit einer genau definierten Schnittstelle zur Verfügung. Ein DOM-Objektmödell beschreibt unter anderem die Beziehung der einzelnen Objektklassen zueinander, welche und wie viel Unterklassen diese haben dürfen und ob die Eigenschaften wie Attribute oder Elementinhalte verändert werden dürfen. Die Objektklassen entsprechen dabei einem bestimmten Knotentyp und stellen Methoden zur Verfügung, mit denen die Attribute und die Namen eines Elementes festgestellt oder manipuliert werden können. ¹

4.1.1 Standarisierungen von DOM


DOM Level 0: Ist keine W3C-Spezifikation. Es ist lediglich eine Definition der Funktionalität.

DOM Level 1: Baut auf DOM Level 0 auf und legt den Fokus auf HTML- und XML-Dokument-Modelle.

DOM Core: Definiert das Bewegen in einen DOM-Baum und die Manipulation von Knoten.

DOM HTML: Ist die Erweiterung für den Zugriff auf HTML-Dokumente.

²Siehe (w3schools.com, 2010).
DOM Level 2: Enthält Erweiterungen für DOM-Core für u. a. XML-Namensraum-Unterstützung, für DOM-HTML für u. a. XHTML-Dokumente und definiert neue Elemente:

DOM Level 2 Style: Für das Manipulieren der Style Informationen im Dokument.

DOM Level 2 CSS: Für die Manipulation von Layoutinformationen.

DOM Level 2 Views: Definiert den Zugriff auf konkrete Wiedergabearten eines Dokumentes (z. B. Bilder).

DOM Level 2 Events: Definiert ein Eventmodell und standardisiert die Verarbeitung von Ereignissen im Dokument (z. B. Benutzerinteraktion).

DOM Level 2 Traversal und DOM Level 2 Range: Definieren das Durchlaufen des Knotenbaums anhand von bestimmten Auswahlkriterien und das Arbeiten mit Bereichen im Dokument, die bestimmte Elemente und Textknoten umfassen.


DOM Level 3 Core: Spezifiziert den Zugriff und die Manipulation von Inhalt, Struktur und Style in Dokumenten.

DOM Level 3 Events: Erweitert die Funktionalität aus DOM Level 2 Events, um neue Interfaces und neue Events.

DOM Level 3 Views und DOM Level 3 Formatting: Erlauben es, dynamisch auf den Inhalt, Struktur und Style zuzugreifen und diese zu ändern.

DOM Level 3 Load und DOM Level 3 Save: Ermöglicht die Serialisierung von Dokumenten oder Dokumentsteilen sowie das Parsen von XML-Dokumenten in Zeichenketten in Dokument-Objekten.

DOM Level 3 XPath: Erlaubt das Auswählen von Knoten anhand von XPath-Ausdrücken.

DOM Level 3 Validation: Erlaubt das Prüfen, ob nach einer dynamischen Änderung (Hinzufügen oder Entfernen von Knoten) das DOM-Dokument valide bleibt.

4.1.2 DOM Schnittstellen und Knotentypen


```java
private static Document createXml(ResultSet resultSet) {
    DocumentBuilderFactory factory = DocumentBuilderFactory.newInstance();
    DocumentBuilder builder = factory.newDocumentBuilder();
    Document document = builder.newDocument();
    Element root = document.createElement("result");
    document.appendChild(root);

    while (resultSet.next()) {
        Element sentence =
            document.createElement("sentence");
        sentence.setAttribute("id", (resultSet.getString(1)));
        sentence.appendChild(
            document.createTextNode(
                resultSet.getString(2)));
        root.appendChild(sentence);
    }
    return document;
}
```

Listing 4.1: Beispiel der Erstellung eines XML-Dokumentes mittels DOM


Mit `Element.appendChild(sentence)` wird das erzeugte Element dem Wurzelelement angefügt. Dabei ist anzumerken, dass die Beispiele in diesem Kapitel nur einen Einblick über die allgemeine Verwendung von DOM Implementierungen geben sollen, da sie alle ähnlich zu verwenden sind.

---

Das Parsen eines XML-Dokumentes ist im nächsten Listing 4.2 verdeutlicht.

```java
private static String parseDocument(String xmlDocument) {
    ByteArrayInputStream inputStream = new ByteArrayInputStream(xmlDocument.getBytes("UTF-8"));
    DocumentBuilderFactory factory = DocumentBuilderFactory.newInstance();
    DocumentBuilder builder = factory.newDocumentBuilder();
    Document document = builder.parse(inputStream);
    NodeList sentences = document.getElementsByTagName("sentence");
    for (int i = 0; i < sentences.getLength(); i++) {
        ...
    }
    ...
}
```

Listing 4.2: Beispiel des Parsens eines XML-Dokumentes mittels DOM


Im nachfolgenden Abschnitt dieses Kapitels werden verschiedene DOM-Implementierungen vorgestellt. Hierbei ist noch zu erwähnen, dass sich die Benutzung der Implementierungen sehr ähnlich sind und keine nennenswerten Unterschiede aufweisen.

### 4.1.3 Implementierungen von DOM


**JDOM**

Somit kann JDOM auf die Features und Vorteile, wie zum Beispiel Collections, Reflection oder Methodenüberladung, der Sprache zurückgreifen. Die JDOM Bibliothek besteht dabei aus sechs Paketen, die in der nachfolgenden Tabelle kurzen erläutert werden.

<table>
<thead>
<tr>
<th>Paket</th>
<th>Beschreibung</th>
</tr>
</thead>
<tbody>
<tr>
<td>org.jdom</td>
<td>Beinhaltet Klassen, die die verschiedenen Knotentypen repräsentieren.</td>
</tr>
<tr>
<td>org.jdom.input</td>
<td>Beinhaltet Klassen, um Dokumente zu erstellen.</td>
</tr>
<tr>
<td>org.jdom.output</td>
<td>Beinhaltet Klassen, um XML-Dokumente auszulesen.</td>
</tr>
<tr>
<td>org.jdom.transform</td>
<td>Beinhaltet Klassen, für XSLT-Transformationen.</td>
</tr>
<tr>
<td>org.jdom.xpath</td>
<td>Beinhaltet Klassen, um mittels XPath zu operieren.</td>
</tr>
<tr>
<td>org.jdom.adapters</td>
<td>Beinhaltet Helper-Klassen für DOM.</td>
</tr>
</tbody>
</table>

Tabelle 4.1: JDOM-Komponenten

JDOM stellt dabei einige Parser, wie z.B. SAXOutputter und DOMOutputter sowie verschiedene Builder, wie z.B. DOMBuilder und SAXBuilder, für das effiziente Einlesen und Generieren von XML-Dokumenten bereit.4

DOM4J

Ebenfalls wie JDOM ist Document Object Model for Java (DOM4J) eine frei verfügbare Javabibliothek, um XML zu verarbeiten und wurde ausschließlich für die Sprache Java entwickelt.5

XOM


4 Vgl. (Hunter, 2002).
5 Vgl. (Hari, 2009).
6 Vgl. (Harold, 2009).

4.2 Streaming API for XML


### 4.2.1 Pull Parsing und Push Parsing im Vergleich


### 4.2.2 StAX-API


---

7 Vgl. (Harold, 2003) und (Oracle, 2010).

Im nachfolgenden Listing 4.3 ist ein Beispiel für die Verwendung der StAX-Cursor-API gegeben. Dabei wurde das Package `javax.xml.stream` für die Erstellung der Beispiele verwendet.

```java
private static void createXml(ResultSet resultSet, OutputStream outputStream) {
    XMLOutputFactory outputFactory = XMLOutputFactory.newInstance();
    XMLStreamWriter streamWriter = outputFactory.createXMLStreamWriter(outputStream);
    streamWriter.writeStartDocument();
    streamWriter.writeStartElement("result");

    while (resultSet.next()) {
        streamWriter.writeStartElement("sentence");
        streamWriter.writeAttribute("id", (new Integer(resultSet.getRow() + startId - 1).toString()));
        streamWriter.writeCharacters(resultSet.getString(2));
        streamWriter.writeEndElement();
    }

    streamWriter.writeEndElement();
    streamWriter.writeEndDocument();
}
```

Listing 4.3: Beispiel der Erstellung eines XML-Dokumentes mittels StAX und der Cursor-API


Das Parsen eines XML-Dokumentes ist im Listing 4.4 gezeigt.

---

8Die Beispiele wurden hier lediglich mit der StAX-Cursor-API entwickelt, da nur diese in den Tests verwendet wurde. Die StAX-Iterator-API soll an dieser Stelle nur erwähnt bleiben.
private static String parseDocument(String xmlDocument) {
    XMLInputFactory inputFactory = XMLInputFactory.newInstance();
    StringReader reader = new StringReader(response);
    XMLStreamReader streamReader = inputFactory.createXMLStreamReader(reader);

    for (int event = streamReader.next();
         event != XMLStreamConstants.END_DOCUMENT;
         event = streamReader.next()) {
        if (event == XMLStreamConstants.START_ELEMENT
                && streamReader.getLocalName().equals("sentence")
            ) {
            int sentenceId = Integer.parseInt(streamReader.getAttributeValue(0));
        }
    }
    ... 
}

Listing 4.4: Beispiel des Parsens eines XML-Dokumentes mittels StAX Cursor API


Vergleich der Iterator- und Cursor-API


4.2.3 Anwendungsgebiete von StAX

Die StAX-Spezifikation findet in zahlreichen Gebieten ihren Einsatz, welche nachfolgen exemplarisch aufgezählt werden.

Data Binding

---

9 (Oracle, 2010)
10 (Oracle, 2010)
11 Aus (Oracle, 2010)
KAPITEL 4. PROGRAMMIERSCHNITTSTELLEN FÜR XML

- Unmarshalling eines XML-Dokumentes.
- Marshalling eines XML-Dokumentes.
- Parallele Dokumentenverarbeitung.

SOAP-Nachrichtenverarbeitung

- Parsen simpler, planbarer Strukturen.
- Parsen von Graphrepräsentationen mit Vorwärtsrichtung.
- Parsen von WSDL.

virtuelle Datenquellen

- Betrachtet als XML-Daten, gespeichert in Datenbanken.
- Betrachte Daten in Java-Objekten, generiert durch Data Binding.
- Navigiert durch einen DOM-Baum, als Stream von Events.

Parsen spezifischer XML-Vokabulare.

4.2.4 StAX Implementierungen

SJSXP


Javolution


Woodstox und StAX RI

Woodstox ist ein von der Firma Codehaus frei verfügbarer, validierender StAX Parser. Er wurde mit Java entwickelt und basiert auf der Spezifikation JSR-173. Streaming API for XML Reference Implementation (StAX RI) wurde ebenfalls von der Firma Codehaus entwickelt und ist die Referenzimplementierung von StAX.

---

4.3 Performanceevaluation der XML-Schnittstellen

Im diesem Abschnitt werden die Testergebnisse zum Vergleich der Programmierschnittstellen für XML ausgewertet, um festzustellen, welche Schnittstelle die bessere Performance besitzt. Für die Ermittlung, wurden 1.000 Requests pro Nachrichtengröße, die zwischen 1, 100, 1.000, 10.000 und 100.000 Elementen variierte, gestellt. Die XML-Programmierschnittstellen wurden in einer REST-basierten Umgebung implementiert. Dafür wurden acht verschiedene Webanwendungen erstellt, von denen jede eine der bereits erwähnten Implementierungen der DOM- bzw. StAX-Spezifikationen verwendet.

Service-Implementierung


Alle untersuchten Implementierungen der Spezifikationen von DOM und StAX bauen auf Parser auf. Theoretisch könnten diese mit allen zur Verfügung stehenden Parseern, wie z.B. Piccolo, Xerces, Crimson, usw., laufen. Für die repräsentative Aussage der Tests wurde für jede getestete Implementierung Xerces als Parser verwendet\(^\text{15}\).

Versuchsergebnisse

In diesem Abschnitt werden die erzielten Resultate präsentiert und ausgewertet. Die Gesamtverarbeitungszeit wurden dabei auf der Clientseite gemessen und spiegelt die Zeit vom Versenden des Requests bis hin zum Empfang des Response vom Servers wieder. Die Serververarbeitungszeit ist die Zeit, die der Server zur Verarbeitung einer Anfrage und zur Generierung einer Antwort benötigt.

In den Tabellen 4.2 und 4.3 sind die Ergebnisse der Testdurchläufe tabellarisch aufgelistet.

\begin{table}[h]
\centering
\begin{tabular}{|c|c|c|c|c|c|}
\hline
 & & Serververarbeitungszeit pro Anzahl Elemente & & & \\
 & 1 & 100 & 1.000 & 10.000 & 100.000 \\
\hline
DOM & 4 & 10 & 27 & 233 & 2384 \\
 & 2 & 6 & 25 & 198 & 2480 \\
DOM4J & 3 & 7 & 23 & 204 & 2461 \\
 & 3 & 8 & 29 & 297 & 2811 \\
\hline
StAX & 2 & 6 & 20 & 179 & 1773 \\
 & 3 & 7 & 24 & 187 & 2230 \\
StAX RI & 2 & 6 & 21 & 179 & 2040 \\
 & 2 & 5 & 19 & 172 & 2048 \\
\hline
\end{tabular}
\caption{XML-Schnittstellen Verarbeitungszeiten auf dem Server}
\end{table}

Wie schon bereits in den vorherigen Abschnitten vermutet, besteht zwischen kleineren Dokumenten kein signifikanter Unterschied in der Gesamtverarbeitungszeit der einzelnen XML-Schnittstellen. Erst bei größeren Dokumenten ab 1.000 Elementen kristallisiert sich heraus,\(^\text{15}\) Eine Untersuchung der Performance der verschiedenen Parser wurde in (Büchler, 2005) durchgeführt.
KAPITEL 4. PROGRAMMIERSCHNITTSTELLEN FÜR XML

Tabelle 4.3: XML-Schnittstellen Gesamtverarbeitungszeiten

<table>
<thead>
<tr>
<th></th>
<th>Gesamtverarbeitungszeit pro Anzahl Elemente</th>
</tr>
</thead>
<tbody>
<tr>
<td></td>
<td>1</td>
</tr>
<tr>
<td><strong>DOM</strong></td>
<td></td>
</tr>
<tr>
<td>DOM</td>
<td>8</td>
</tr>
<tr>
<td>JDOM</td>
<td>6</td>
</tr>
<tr>
<td>DOM4J</td>
<td>6</td>
</tr>
<tr>
<td>XOM</td>
<td>6</td>
</tr>
<tr>
<td><strong>StAX</strong></td>
<td></td>
</tr>
<tr>
<td>Javolution</td>
<td>6</td>
</tr>
<tr>
<td>SJSXP</td>
<td>6</td>
</tr>
<tr>
<td>StAX RI</td>
<td>6</td>
</tr>
<tr>
<td>Woodstox</td>
<td>5</td>
</tr>
</tbody>
</table>

dass DOM-Implementierungen längere Zeit für die Erstellung der Ergebnisdokumente benötigen als die StAX-Implementierungen. Negativ hervorzuheben ist in diesem Test XOM, da diese relativ viel Zeit für die Bearbeitung der Anfragen benötigt. Am schnellsten verarbeitet die Webanwendung die Anfragen als Woodstox zur Generierung von XML verwendete wurde.

4.4 Zusammenfassung

In diesem Kapitel wurde StAX mit DOM verglichen. Hier ist zu erwähnen, dass die Integration und Verwendung aller Implementierungen sehr einfach war. Die Entscheidung über die Verwendung von DOM oder StAX, hängt von der Größe der zu generierenden Dokumente und der zur Verfügung stehenden Hardware ab. So lange die Dokumente klein genug sind\textsuperscript{16}, gibt es keinen signifikanten Unterschied zwischen den beiden Programmierschnittstellen. Erst wenn diese größer werden\textsuperscript{17}, ist StAX zu bevorzugen, da hier aufgrund des geringeren Speicherverbrauches und der schnelleren Verarbeitung die bessere Performance besteht.

\textsuperscript{16}Bis 100 Elemente.
\textsuperscript{17}Ab 1.000 Elemente.
5 XML Repräsentationen


5.1 Arten von Markup

5.1.1 Inline Markup


Bei der Erstellung eines Dokumentes im Inline-Format existieren zwei Lösungsansätze. Der Erste verwendet ausschließlich Attribute, um weitere Informationen einem Objekt hinzuzufügen, was im Codebeispiel 5.1 ersichtlich ist. Es stellt einen Satz dar, der zu jedem enthaltenen Wort die Frequenz angibt. Das Dokument ist am Beispiel des Wortes Follow folgendermaßen zu lesen: „Das Wort Follow mit der Id 17.185 des Satzes mit der Id 3 hat die Frequenz 5“.

```
<?xml version="1.0" encoding="UTF-8"?>
<result>
  <sentence s_id="3">
    <token frequency="5" w_id="17185">Follow</token>
    <token frequency="160837" w_id="1">the</token>
    <token frequency="113" w_id="2213">colors</token>
    <token frequency="94352" w_id="2">of</token>
    <token frequency="160837" w_id="1">the</token>
    <token frequency="39" w_id="4982">Nineteenth</token>
  </sentence>
</result>
```

1Detaillierte Beschreibung der Servicekette Siehe Kapitel 5.1.3.
KAPITEL 5. XML REPRÄSENTATIONEN

Möglichkeit zwei ist im Listing 5.2 zu sehen. Dieses Beispiel ist eine gekürzte Variante für die ersten beiden Wörter des Satzes. Es hat die gleiche Aussagekraft wie das erste Beispiel, jedoch werden hier die Informationen über die Frequenz eines Wortes als Element und nicht als Attribut gespeichert.

```
<?xml version="1.0" encoding="UTF-8" ?>
<result>
  <sentence s_id="3">
    <frequency count="5">
      <token w_id="17185">Follow</token>
    </frequency>
    <frequency count="160837">
      <token w_id="1">the</token>
    </frequency>
    <frequency count="113">
      ...
    </frequency>
    ...
  </sentence>
</result>
```

Listing 5.2: Inline Markup mit Elementen

In diesen beiden vorangegangenen Beispielen ist somit sichtbar, dass die Angabe von Strukturinformationen in Elementen bzw. in Attributen geschehen kann. Dies hat erhebliche Auswirkungen auf die Größe und somit auf die Geschwindigkeit des Übertragens und des Parsens des resultierenden Dokumentes.

5.1.2 Stand-Off Markup

Stand-Off- oder externes Markup wird verwendet, wenn das Markup außerhalb des Textes eingesetzt werden soll. Die Gründe hierfür könnten beispielsweise sein, dass keine Schreibbreche auf die Daten bestehen oder die Daten beinhalten bereits Markup, das jedoch inkompatibel zu dem Markup ist, mit dem das Dokument erweitert werden soll. Es besteht allerdings die Möglichkeit jedes externe Markup in internes umzuwandeln, indem einfach das externe Markup direkt in den Text hinzugefügt und die Referenzen auf die Objekte mit den eigentlichen Objekten ersetzt werden.

generiert werden. Somit können inkompatibles Markup und fehlende Schreibrechte ausgeschlossen werden. Das nachfolgende Codebeispiel 5.3 zeigt einen Ausschnitt aus einem Dokument im Stand-Off Format.

```xml
<?xml version="1.0" encoding="UTF-8"?>
<result>
  <sentence s_id="3"> Follow the colors of the Nineteenth. </sentence>
  <sentence s_id="3"
  <tokens>
    <token w_id="4982" s_id="3">Nineteenth</token>
    <token w_id="1" s_id="3">the</token>
    <token w_id="2213" s_id="3">colors</token>
    ...
  </tokens>
  <frequencies>
    <frequency w_id="4982">39</frequency>
    <frequency w_id="1">160837</frequency>
    <frequency w_id="2213">113</frequency>
    ...
  </frequencies>
</result>
```

Listing 5.3: Stand-Off Markup


### 5.1.3 Performanceevaluation der Markup-Arten Inline und Stand-Off


---
² Siehe http://weblicht.sfs.uni-tuebingen.de/.
³ http://www.clarin.eu/external/
⁴ Auch in anderen Bereichen ist das sogenannte Service-Chaining weit verbreitet.
Performancetests gewählt.


![Abbildung 5.1: Aufruf einer Kette von Services durch den Client](image)


Weiterhin waren beim Vergleich der beiden Markup-Arten vor allem die Kriterien Größe der Dokumente, Zeit vom Senden des Requests bis hin zum Empfangen des Response vom Server sowie der Schwierigkeitsgrad der Entwicklung relevant.

**Service-Implementierung**

Bei der Implementierung des Testszenarios wurden drei RESTful Services entwickelt, die die Dokumente mithilfe von StAX generierten.5 Für die Simulation der wechselseitigen Client-Server-Interaktion musste auf der Serverseite ein Simulator entwickelt werden, der nacheinander die Services auf rief. Die ersten beiden Services konvertierten dabei das XML-Dokument in eine Zeichenkette. Der FrequencyService war dabei der letzte Service in der Kette. Er übergab eine **OutputStream**-Instanz, die das endgültige Dokument beinhaltete. Bei der Erstellung des Ergebnisdokumentes wurde in jedem Service und auch bei jeder Markup-Variante eine **BufferedWriter**-Instanz an eine **XMLStreamWriter**-Instanz übergeben, welche schließlich das Dokument erzeugte. Die größe der Puffers der BufferedWriter-Instanz wurde

---

5Dabei wurde SJSXLP verwendet.


**Versuchsergebnisse**

Es wurden sechs verschiedene Nachrichtengrößen mit 1-, 100, 1.000, 10.000, 100.000 und 200.000 Elementen mit jeweils 1.000 Requests getestet. Die Zeiten wurden auf der Clientseite gemessen und spiegeln die Zeit vom Senden des Requests bis zum Empfang des Response wieder.

---

6Zum Zeitpunkt der Entwicklung der Services schien eine Datenbankabfrage pro existierendem Satz im Dokument die effektivere Variante, weshalb diese auch entwickelt wurde.

Neben den Zeiten (Siehe Tabelle 5.2) wurden auch die Größen der Nachrichten untersucht, die in Tabelle 5.1 dargestellt sind. In den Abbildungen 5.2 und 5.3 wurden die Ergebnisse normalisiert und aufgrund der besseren Übersichtlichkeit in einem Diagramm veranschaulicht.

<table>
<thead>
<tr>
<th>Anzahl der Elemente im Dokument</th>
<th>1</th>
<th>100</th>
<th>1.000</th>
<th>10.000</th>
<th>100.000</th>
<th>200.000</th>
<th>400.000</th>
<th>500.000</th>
</tr>
</thead>
<tbody>
<tr>
<td><strong>Inline mit Elementen</strong></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>2</td>
<td>122</td>
<td>1.093</td>
<td>14.138</td>
<td>111.893</td>
<td>211.129</td>
<td>436.257</td>
<td>543.039</td>
<td></td>
</tr>
<tr>
<td><strong>Inline mit Attributen</strong></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>2</td>
<td>92</td>
<td>823</td>
<td>10.623</td>
<td>84.292</td>
<td>159.041</td>
<td>328.435</td>
<td>408.834</td>
<td></td>
</tr>
<tr>
<td><strong>Stand-Off</strong></td>
<td>3</td>
<td>146</td>
<td>1.191</td>
<td>13.703</td>
<td>104.692</td>
<td>195.994</td>
<td></td>
<td></td>
</tr>
</tbody>
</table>

Tabelle 5.1: Nachrichtengröße der Markup-Arten

Bei der Auswertung des Vergleichs der Nachrichtengröße ist aufgefallen das keinere Dokumente im Stand-Off Format bis zu 60% größer sind als Dokumente im Inline-Format. Erst bei einer höheren Anzahl an Elementen im Dokument wird dieses verhältnismäßig kleiner. Der Grund dafür ist die Arbeitsweise des FrequencyService. Denn dieser beinhaltet für Dokumente im Stand-Off-Format, jedes Wort das in den abgefragten Sätzen vorkommt

\[7\] Aus den eingangs erwähnten Gründen konnten die Dokumente mit Stand-Off Markup nur bis 300.000 und die Dokumente im Inline-Format bis 500.000 Elementen generiert werden.
genau einmal. Während bei Dokumenten mit Inline-Markup jedes Wort mit den Frequenz Informationen versehen wird, egal wie oft es in den vorherigen Sätzen enthalten war. Tabelle 5.2 und die dazugehörige Abbildung 5.3 zeigen die durchschnittliche Gesamtverarbeitungszeit und die Normalisierung der gemessenen Zeiten.

<table>
<thead>
<tr>
<th>Anzahl der Elemente im Dokument</th>
<th>1</th>
<th>100</th>
<th>1.000</th>
<th>10.000</th>
<th>100.000</th>
<th>200.000</th>
</tr>
</thead>
<tbody>
<tr>
<td>Inline mit Elementen</td>
<td>9</td>
<td>148</td>
<td>1.277</td>
<td>13.514</td>
<td>118.197</td>
<td>238.446</td>
</tr>
<tr>
<td>Inline mit Attributen</td>
<td>10</td>
<td>147</td>
<td>1.274</td>
<td>13.469</td>
<td>114.887</td>
<td>223.313</td>
</tr>
<tr>
<td>Stand-Off</td>
<td>12</td>
<td>76</td>
<td>577</td>
<td>6.722</td>
<td>99.667</td>
<td>274.143</td>
</tr>
</tbody>
</table>

Tabelle 5.2: Gesamtverarbeitungszeit der Markup-Arten

Abbildung 5.3: Durchschnittliche Gesamtverarbeitungszeit der Markup-Arten pro Element

dafür ist, dass beim Inline-Markup mit Attributen nicht so viel Zeichen für die Zusatzinformationen verwendet werden, wie beim Inline-Markup mit Elementen. Aus diesem Grund sind auch die Zeiten der Request Response-Zyklen bis zu 15% geringer, da hier das Netzwerk nicht so sehr belastet wird und der Client weniger Bytes empfangen muss.

Eines ist noch anzumerken: die durchgeführten Testreihen für die Dokumente mit 400.000 und 500.000 Elementen sind nicht repräsentativ, da JMeter hier einen sehr hohen Speicherverbrauch beim Empfang der Nachrichten hatte und Java den kompletten zur Verfügung stehenden Speicher belegte. Hier wurde teilweise mehr Zeit für den Empfang der Nachricht aufgewendet, als für die Generierung und die Übertragung.

Aufgrund der unterschiedlichen Resultate der beiden Varianten der Dokumente mit Inline-Markup wurden zusätzlich die Zeiten des Parsens der Dokumente auf der Seite des Clients betrachtet. Diese sind in Tabelle 5.3 gezeigt.

<table>
<thead>
<tr>
<th>Anzahl der Elemente im Dokument</th>
<th>1</th>
<th>100</th>
<th>1.000</th>
<th>10.000</th>
<th>100.000</th>
</tr>
</thead>
<tbody>
<tr>
<td>Inline mit Elementen</td>
<td>0,4</td>
<td>4,6</td>
<td>37,8</td>
<td>462,3</td>
<td>3663,8</td>
</tr>
<tr>
<td>Inline mit Attributen</td>
<td>0,5</td>
<td>3,8</td>
<td>30,8</td>
<td>393,7</td>
<td>3115,3</td>
</tr>
</tbody>
</table>

Tabelle 5.3: Zusammenfassung der Inline Markup Parsingergebnisse


```xml
<?xml version="1.0" encoding="UTF-8" ?>

    ...
    if (event != XMLStreamConstants.START_ELEMENT)
    {
        continue;
    }
    
    if (StreamReader.getLocalName().equals("frequency"))
    {
        StreamReader.next();
        frequency = StreamReader.getAttributeValue(0);
        continue;
    }
    
    if (StreamReader.getLocalName().equals("token"))
    {
        StreamReader.next();
    }
```
5.2 Repräsentation von XML mittels TEI


---

8 Standard Generalized Markup Language ist eine Metasprache, mit deren Hilfe verschiedene Markup-Sprachen (wie XML) für Dokumente definiert werden.
KAPITEL 5. XML REPRÄSENTATIONEN


5.2.1 Einführung TEI

In dieser Einführung wurde die Sprache TEI Version 5 verwendet. Das Bereitstellen eines Vokabulars zur Beschreibung von Texten, ist die Philosophie von TEI. In TEI (Version 5) stehen 22 Module zur Verfügung, die in der Tabelle 5.4 aufgelistet sind.

<table>
<thead>
<tr>
<th>Modul</th>
<th>Beschreibung</th>
</tr>
</thead>
<tbody>
<tr>
<td>analysis</td>
<td>Beinhaltet analytische Mechanismen.</td>
</tr>
<tr>
<td>certainty</td>
<td>Gewiss und Ungewiss.</td>
</tr>
<tr>
<td>core</td>
<td>Beinhaltet Grundelemente von TEI.</td>
</tr>
<tr>
<td>corpus</td>
<td>Erweiterungen für den Header für Korpora.</td>
</tr>
<tr>
<td>declarefs</td>
<td>System-Deklarationen.</td>
</tr>
<tr>
<td>dictionaries</td>
<td>Gedruckte Wörterbücher.</td>
</tr>
<tr>
<td>drama</td>
<td>Drehbücher.</td>
</tr>
<tr>
<td>figures</td>
<td>Tabellen, Formulare, Abbildungen.</td>
</tr>
<tr>
<td>gaiji</td>
<td>Dokumentation von Buchstaben und Glyphen.</td>
</tr>
<tr>
<td>header</td>
<td>Header von TEI.</td>
</tr>
<tr>
<td>iso-fs</td>
<td>Strukturen.</td>
</tr>
<tr>
<td>linking</td>
<td>Ausrichtung, Segmentierung und Verlinkung.</td>
</tr>
<tr>
<td>msdescription</td>
<td>Beschreibung von Manuskripten.</td>
</tr>
<tr>
<td>namesdates</td>
<td>Namen und Daten.</td>
</tr>
<tr>
<td>nets</td>
<td>Graphen, Netzwerke und Bäume.</td>
</tr>
<tr>
<td>spoken</td>
<td>Transkripte Sprache.</td>
</tr>
<tr>
<td>tagdocs</td>
<td>Dokumentation von TEI-Modulen.</td>
</tr>
<tr>
<td>tei</td>
<td>Deklaration von Datentypen, Klassen und Makros.</td>
</tr>
<tr>
<td>textcrit</td>
<td>Textkritiken.</td>
</tr>
<tr>
<td>textstructure</td>
<td>standardmäßige Textstruktur.</td>
</tr>
<tr>
<td>transcr</td>
<td>Transkription von primären Quellen.</td>
</tr>
<tr>
<td>verse</td>
<td>Struktur von Versen.</td>
</tr>
</tbody>
</table>

Tabelle 5.4: Module in TEI P5

Jedes TEI-Dokument besteht aus einem Kopf, dem teiHeader, der das Titelblatt darstellt, sowie einem Textkörper, text, der den elektronischen Textinhalt enthält. Diese Struktur ist in

9Vgl. (Georg Braungart, 2009).
Abbildung 5.6 verdeutlicht\textsuperscript{10}.

```xml
<?xml version="1.0" encoding="UTF-8"?>
<TEI>
  <teiHeader>
    ...
  </teiHeader>
  <text>
    ...
  </text>
</TEI>
```

Listing 5.6: Grundstruktur eines TEI-Dokumentes

Texte lassen sich in TEI auf zwei Möglichkeiten gruppieren. Wenn die Texte alle den gleichen Header verwenden können, wäre folgende Struktur für das Dokument möglich:

```xml
<?xml version="1.0" encoding="UTF-8"?>
<TEI>
  <teiHeader>
    ...
  </teiHeader>
  <text>
    <group>
      <text>
        ...
      </text>
    </group>
  </text>
</TEI>
```

Listing 5.7: Gruppierung von Texten mit gleichen Header in TEI

Bei Texten, die einen eigenen Header verwenden, sieht die Dokumentation vor, dass die Elemente nochmals unterteilt werden.

```xml
<?xml version="1.0" encoding="UTF-8"?>
<TEICorpus>
  <teiHeader>...</teiHeader>
  <TEI>...</TEI>
  <TEI>...</TEI>
  ...
</TEICorpus>
```

Listing 5.8: Gruppierung von Texten mit unterschiedlichen Header

\textsuperscript{10}Es wird hier nur auf die nötigsten Elemente von TEI eingegangen, da die Erläuterung aller Elemente den Rahmen dieser Arbeit sprengen würde.
In Listing 5.7 ist verdeutlicht, dass das text-Element mehrfach auftaucht. In der obersten Hierarchie fasst das text-Element alles außerhalb des Headers zusammen. In der unteren Hierarchie werden die einzelnen Texte mittels des group-Elementes gebündelt. Diese Form der Formatierung findet häufig bei Werken eines Autors statt. In Listing 5.8 hat jedes TEI-Element seinen eigenen Header.

In TEI ist es möglich, den Text zu untergliedern. So kann beispielsweise bei der Wiedergabe eines Buches mit den Elementen <front>, <body> und <back> dieses unterteilt werden. Front würde das Titelblatt, body den eigentlichen Text und back das Stichwortverzeichnis oder Glossar des Buches enthalten. Mit dem Element <div> würden dann die einzelnen Kapitel dargestellt. Attribute im div-Element beschreiben die Art der Untergliederung, was im folgenden Beispiel 5.9 zu sehen ist.

```xml
...<text>
  <body>
    <div type="Fragment" n="t">
      <ab>
        <lb n="1t"/>
        <w n="TLG\_wID\_9219">
          <note type="Frequency">16</note>
          constituting
        </w>
      </ab>
    </div>
...  </body>
</text>
...
```

Listing 5.9: Untergliederung von Texten

Zusätzlich bietet TEI die Möglichkeit, mit nummerierten und unnummerierten div-Elementen zu arbeiten. Die einfachere Verwaltung ist bei nummerierten Elementen vorteilhaft. Die Schachtelungstiefe ist jedoch begrenzt. TEI bietet nur die Möglichkeit mit acht Hierarchien\textsuperscript{11} zu arbeiten. Unnummerierte div-Elemente können dahingegen beliebig tief geschachtelt werden, sind aber dafür schwerer zu verwalten.

Für die Kodierung von Zeilenumbrüchen und Seitenwechsel gibt TEI mit <lb> für linebreak und <pb> für pagebreak zwei weitere Elemente vor.


```xml
<teiHeader>
  <fileDesc></fileDesc>
  <encodingDesc></encodingDesc>
  <profileDesc></profileDesc>
  <revisionDesc></revisionDesc>
```

\textsuperscript{11}Von div0 bis div7.
Das Element `fileDesc` beinhaltet bibliografische Angaben zum im Textkörper enthaltenen Text. In `encodingDesc` werden Beziehungen zwischen dem Text und der Vorlage dokumentiert. `profilDesc` enthält unter anderem Angaben zur verwendeten Sprache oder zur Entstehung des Textes. In `revisionDesc` kann schließlich die Überarbeitungsgeschichte des Textes dokumentiert werden.

### 5.2.2 Markup-Arten in TEI


### 5.2.3 Performanceevaluation der Markup-Arten in TEI

In diesem Abschnitt werden die Tests der Performance der Markup-Arten, Inline und Stand-Off, in der TEI-Syntax dargestellt. ¹²

### 5.2.4 Versuchsergebnisse

In den folgenden Tabellen 5.5, und 5.6 werden die Ergebnisse der Versuchsreihen präsentiert.

<table>
<thead>
<tr>
<th>Anzahl der Elemente</th>
<th>Gesamtverarbeitungszeit (in ms)</th>
<th>Durchsatz (in Requests/Sekunde)</th>
<th>Nachrichtengröße (in KB)</th>
</tr>
</thead>
<tbody>
<tr>
<td>1</td>
<td>9</td>
<td>67,5</td>
<td>2,24</td>
</tr>
<tr>
<td>100</td>
<td>118</td>
<td>7,9</td>
<td>117</td>
</tr>
<tr>
<td>1.000</td>
<td>1.053</td>
<td>0,9</td>
<td>1.061</td>
</tr>
<tr>
<td>10.000</td>
<td>12.146</td>
<td>0,08</td>
<td>13.615</td>
</tr>
<tr>
<td>100.000</td>
<td>115.022</td>
<td>0,01</td>
<td>108.720</td>
</tr>
</tbody>
</table>

KAPITEL 5. XML REPRÄSENTATIONEN

<table>
<thead>
<tr>
<th>Anzahl der Elemente</th>
<th>Gesamtverarbeitungszeit (in ms)</th>
<th>Durchsatz (in Requests/Sekunde)</th>
<th>Nachrichtengröße (in KB)</th>
</tr>
</thead>
<tbody>
<tr>
<td>1</td>
<td>9</td>
<td>98,5</td>
<td>3,23</td>
</tr>
<tr>
<td>100</td>
<td>92</td>
<td>10,7</td>
<td>152</td>
</tr>
<tr>
<td>1.000</td>
<td>625</td>
<td>1,6</td>
<td>1,177</td>
</tr>
<tr>
<td>10.000</td>
<td>6.729</td>
<td>0,15</td>
<td>12,262</td>
</tr>
<tr>
<td>100.000</td>
<td>96,342</td>
<td>0,01</td>
<td>91,005</td>
</tr>
</tbody>
</table>

Tabelle 5.6: Versuchsergebnisse der Loadtests von TEI mit Stand-Off-Markup


In den Abbildungen 5.4 und 5.5 wurde zur Verdeutlichung der Ergebnisse, die Größe der Nachrichten sowie die Gesamtverarbeitungszeit normalisiert und dargestellt.

![Abbildung 5.4: Durchschnittliche Nachrichtengröße pro Element](image)


5.3 Zusammenfassung

In diesem Kapitel wurden die beiden Markup-Arten Inline und Stand-Off erläutert und miteinander verglichen. Das Ergebnis dieses Vergleiches ist, dass die Entwicklung von Services,

Diese Punkte zeigen, dass es besser ist mehr Aufwand in die Entwicklung der Services zu investieren, so dass diese Dokumente mit Inline-Markup generieren. Das schont Ressourcen sowohl auf der Seite des Clients als auch auf der Seite des Servers.
6 Fazit

CXF weitere nennenswerte Frameworks gibt die nicht Teil dieser Untersuchung waren. Weiterhin wurden, aufgrund der Aufgabenstellung, lediglich einfache Services die Datenbank-Lookups ausführen getestet und somit das Manipulieren oder Löschen von Datensätzen nicht mit betrachtet.


---

1 Es wurde mehrfach experimentiert, um dieses Problem zu umgehen, jedoch konnte das Intervall minimal auf sekündlich gestellt werden, was jedoch den Server zu sehr belastete und somit die Ergebnisse sehr beeinflusste.


A Anhang

A WSDL Dateien

A.1 Apache Axis 1

```xml
<?xml version="1.0" encoding="UTF-8" ?>
<wsdl:definitions targetNamespace="http://ServiceTypes"
    xmlns:apachesoap="http://xml.apache.org/xml-soap"
    xmlns:impl="http://ServiceTypes"
    xmlns:intf="http://ServiceTypes"
    xmlns:wsdl="http://schemas.xmlsoap.org/wsdl/
    xmlns:wsdlsoap="http://schemas.xmlsoap.org/wsdl/soap/
    xmlns:xsd="http://www.w3.org/2001/XMLSchema">

<wsdl:types>
    <schema elementFormDefault="qualified"
        targetNamespace="http://ServiceTypes"
        xmlns="http://www.w3.org/2001/XMLSchema">
        <element name="getSentences">
            <complexType>
                <sequence>
                    <element name="startId" type="xsd:int" />
                    <element name="endId" type="xsd:int" />
                </sequence>
            </complexType>
        </element>
        <element name="getSentencesResponse">
            <complexType>
                <sequence>
                    <element maxOccurs="unbounded"
                        name="getSentencesReturn"
                        type="xsd:string" />
                </sequence>
            </complexType>
        </element>
    </schema>
</wsdl:types>
<wsdl:message name="getSentencesRequest">
```
Listing A.1: Automatisch generierte WSDL Datei von Apache Axis 1

A.2 Apache Axis 2
<wSDL:documentation>Please Type your service description here</wSDL:documentation>

<wSDL:types>
  <xs:schema xmlns:ax22="http://ServiceTypes"
            attributeFormDefault="qualified"
            elementFormDefault="qualified"
            targetNamespace="http://sql.java/xsd">
    <xs:import namespace="http://sql.java/xsd"/>
    <xs:complexType name="SQLException">
      <xs:complexContent>
        <xs:extension base="ax22:Exception">
          <xs:sequence>
            <xs:element minOccurs="0" name="SQLState"
                       nillable="true" type="xs:string"/>
            <xs:element minOccurs="0" name="errorCode" type="xs:int"/>
            <xs:element minOccurs="0" name="nextException"
                       nillable="true"
                       type="ax21:SQLException"/>
          </xs:sequence>
        </xs:extension>
      </xs:complexContent>
    </xs:complexType>
  </xs:schema>
</wSDL:types>

<xS:element name="SQLException">
  <xs:complexType>
    <xs:sequence>
      
    </xs:sequence>
  </xs:complexType>
</xs:element>
<xs:element minOccurs="0" name="SQLException" nillable="true" type="ax21:SQLException" />
</xs:sequence>
</xs:complexType>
</xs:element>
<xs:element name="getSentences">
<xs:complexType>
<xs:sequence>
<xs:element minOccurs="0" name="startId" type="xs:int" />
<xs:element minOccurs="0" name="endId" type="xs:int" />
</xs:sequence>
</xs:complexType>
</xs:element>
<xs:element name="getSentencesResponse">
<xs:complexType>
<xs:sequence>
<xs:element maxOccurs="unbounded" minOccurs="0" name="return" nillable="true" type="xs:string" />
</xs:sequence>
</xs:complexType>
</xs:element>
</xs:schema>
</wsdl:types>
<wsdl:message name="getSentencesRequest">
<wsdl:part name="parameters" element="ns:getSentences" />
</wsdl:message>
<wsdl:message name="getSentencesResponse">
<wsdl:part name="parameters" element="ns:getSentencesResponse" />
</wsdl:message>
<wsdl:message name="SQLException">
<wsdl:part name="parameters" element="ns:SQLException" />
</wsdl:message>
<wsdl:portType name="SentencesPortType">
<wsdl:operation name="getSentences">
<wsdl:input message="ns:getSentencesRequest" wsaw:Action="urn:getSentences" />
<wsdl:output message="ns:getSentencesResponse" wsaw:Action="urn:getSentencesResponse" />
<wsdl:fault message="ns:SQLException" name="SQLException" wsaw:Action="urn:getSentencesSQLException" />
</wsdl:operation>
</wsdl:portType>
<wsdl:binding name="SentencesSoap11Binding" type="ns:SentencesPortType">
  <soap:binding transport="http://schemas.xmlsoap.org/soap/http" style="document" />
  <wsdl:operation name="getSentences">
    <soap:operation soapAction="urn:getSentences" style="document" />
    <wsdl:input>
      <soap:body use="literal" />
    </wsdl:input>
    <wsdl:output>
      <soap:body use="literal" />
    </wsdl:output>
    <wsdl:fault name="SQLException">
      <soap:fault use="literal" name="SQLException" />
    </wsdl:fault>
  </wsdl:operation>
</wsdl:binding>

<wsdl:binding name="SentencesSoap12Binding" type="ns:SentencesPortType">
  <soap12:binding transport="http://schemas.xmlsoap.org/soap/http" style="document" />
  <wsdl:operation name="getSentences">
    <soap12:operation soapAction="urn:getSentences" style="document" />
    <wsdl:input>
      <soap12:body use="literal" />
    </wsdl:input>
    <wsdl:output>
      <soap12:body use="literal" />
    </wsdl:output>
    <wsdl:fault name="SQLException">
      <soap12:fault use="literal" name="SQLException" />
    </wsdl:fault>
  </wsdl:operation>
</wsdl:binding>

<wsdl:binding name="SentencesHttpBinding" type="ns:SentencesPortType">
  <http:binding verb="POST" />
  <wsdl:operation name="getSentences">
    <http:operation location="Sentences/getSentences" />
    <wsdl:input>
      <mime:content type="text/xml" part="getSentences" />
    </wsdl:input>
    <wsdl:output>
      <mime:content type="text/xml" part="getSentences" />
    </wsdl:output>
  </wsdl:operation>
</wsdl:binding>
Listing A.2: Automatisch generierte WSDL Datei von Apache Axis 2

A.3 Metro 2.0

```
<?xml version="1.0" encoding="UTF-8" ?>
<definitions xmlns:wsu="http://docs.oasis-open.org/wss/2004/01/oasis-200401-wss-wssecurity-utility-1.0.xsd"
    xmlns:wsp="http://www.w3.org/ns/ws-policy"
    xmlns:wsp1_2="http://schemas.xmlsoap.org/ws/2004/09/policy"
    xmlns:wsam="http://www.w3.org/2007/05/addressing/metadata"
    xmlns:soap="http://schemas.xmlsoap.org/soap/"
    xmlns:tns="http://SoapServices.ServiceTypes/"
    xmlns:xsd="http://www.w3.org/2001/XMLSchema"
    targetNamespace="http://SoapServices.ServiceTypes/"
    name="SentencesService">
    <types>
        <xsd:schema>
            <xsd:import namespace="http://SoapServices.ServiceTypes/"
                schemaLocation="http://localhost:8080/WebServicesKernelMetro2/SentencesService?xsd=1"
            ></xsd:import>
        </xsd:schema>
    </types>
</definitions>
```
Listing A.3: Automatisch generierte WSDL Datei von Metro 2.0
B SOAP-Nachrichten

B.1 Apache Axis 1

<table>
<thead>
<tr>
<th>POST /WebServicesKernelAxis1/services/Sentences HTTP/1.0</th>
</tr>
</thead>
<tbody>
<tr>
<td>Host: localhost:6666</td>
</tr>
<tr>
<td>Content-Type: text/xml; charset=utf-8</td>
</tr>
<tr>
<td>Content-Length: 372</td>
</tr>
<tr>
<td>SOAPAction: &quot;&quot;</td>
</tr>
</tbody>
</table>

```xml
<?xml version='1.0' encoding='UTF-8'?>
<soapenv:Envelope xmlns:xsi="http://www.w3.org/2001/XMLSchema-instance"
                  xmlns:soapenv="http://schemas.xmlsoap.org/soap/envelope/"
                  xmlns:xsd="http://www.w3.org/2001/XMLSchema">
  <soapenv:Body>
    <getSentences xmlns="http://ServiceTypes">
      <startId>1</startId>
      <endId>1</endId>
    </getSentences>
  </soapenv:Body>
</soapenv:Envelope>
```

Listing A.4: Request des Clients

<table>
<thead>
<tr>
<th>HTTP/1.1 200 OK</th>
</tr>
</thead>
<tbody>
<tr>
<td>Server: ApacheCoyote/1.1</td>
</tr>
<tr>
<td>Content-Type: text/xml; charset=utf-8</td>
</tr>
<tr>
<td>Date: Sun, 22 Aug 2010 16:55:34 GMT</td>
</tr>
<tr>
<td>Connection: close</td>
</tr>
</tbody>
</table>

```xml
<?xml version="1.0" encoding="utf-8"?>
<soapenv:Envelope xmlns:soapenv="http://schemas.xmlsoap.org/soap/envelope/"
                  xmlns:xsd="http://www.w3.org/2001/XMLSchema"
                  xmlns:soapenv="http://schemas.xmlsoap.org/soap/envelope/"
                  xmlns:xsi="http://www.w3.org/2001/XMLSchema-instance">
  <soapenv:Body>
    <getSentencesResponse xmlns="http://ServiceTypes">
      <getSentencesResult>Hallo Welt!</getSentencesResult>
    </getSentencesResponse>
  </soapenv:Body>
</soapenv:Envelope>
```

Listing A.5: Response des Servers
B.2 Apache Axis 2

POST /WebServicesKernelAxis2/services/Sentences HTTP/1.1
Content-Type: application/soap+xml; charset=UTF-8; action="urn:getSentences"
User-Agent: Axis2
Host: localhost:6666
Transfer-Encoding: chunked

<?xml version='1.0' encoding='UTF-8'?>
<soapenv:Envelope xmlns:soapenv="http://www.w3.org/2003/05/soap-envelope">
<soapenv:Body>
<ns1:getSentences xmlns:ns1="http://ServiceTypes">
<ns1:startId>1</ns1:startId>
<ns1:endId>1</ns1:endId>
</ns1:getSentences>
</soapenv:Body>
</soapenv:Envelope>

Listing A.6: Request des Clients

HTTP/1.1 200 OK
Server: Apache-Coyote/1.1
Content-Type: application/soap+xml; action="urn:getSentencesResponse"; charset=UTF-8
Transfer-Encoding: chunked
Date: Tue, 06 Apr 2010 16:03:52 GMT

<?xml version='1.0' encoding='UTF-8'?>
<soapenv:Envelope xmlns:soapenv="http://www.w3.org/2003/05/soap-envelope">
<soapenv:Body>
<ns:getSentencesResponse xmlns:ns="http://ServiceTypes"
<ns:return>Hallo Welt!</ns:return>
</ns:getSentencesResponse>
</soapenv:Body>
</soapenv:Envelope>

Listing A.7: Response des Servers

B.3 Metro 2.0

POST /WebServicesKernelMetro2/SentencesService HTTP/1.0
C Markup-Arten für TEI Dokumente

C.1 TEI konformes Beispieldokument mit Inline-Markup

```xml
<?xml version="1.0" ?>
<TEI>
  <teiHeader>...</teiHeader>
```
Listing A.10: Inline-Markup in TEI

C.2 TEI konformes Beispieldokument mit Stand-Off-Markup

```xml
<?xml version="1.0" ?>
<TEI>
  <teiHeader>...</teiHeader>
  <text>
    <body>
      <div type="fragment">
        <ab>
          <lb n="1">
            Hallo Welt!
            <milestone n="TLG_sId_1" unit="sentence">
              Hallo Welt!
            </milestone>
          </lb>
        </ab>
      </div>
      <div type="fragment">
        <ab>
          <lb n="1t">
            Hallo Welt!
            <w n="TLG_wId_113174">Hallo</w>
            <w n="TLG_wId_28662">Welt</w>
          </lb>
        </ab>
      </div>
      <div type="fragment">
        <ab>
          <lb>
            <w n="TLG_wId_113174">Hallo</w>
          </lb>
        </ab>
      </div>
    </body>
  </text>
</TEI>
```
Listing A.11: Stand-Off-Markup in TEI

```xml
<note type="frequency">32</note>
<w>
  <w n="TLG_wId_28662">
    <note type="frequency">177</note>
  </w>
</w>
</text>
</TEI>
```
Literaturverzeichnis


LITERATURVERZEICHNIS


Eidesstattliche Erklärung

“Ich versichere, dass ich die vorliegende Arbeit selbständig und nur unter Verwendung der angegebenen Quellen und Hilfsmittel angefertigt habe, insbesondere sind wörtliche oder sinngemäße Zitate als solche gekennzeichnet. Mir ist bekannt, dass Zuwiderhandlung auch nachträglich zur Aberkennung des Abschlusses führen kann.”

Ort                              Datum                              Unterschrift