

# **Diplomarbeit**

## **Vervollständigung der XML-Schema-Bibliothek XSD4J**

Technische Universität Dresden  
Fakultät Informatik  
Institut für Systemarchitektur  
Professur Rechnernetze

Jiakui Li  
Matrikel Nr.: 3181240

30. April 2010

Hochschullehrer: Prof. Dr. rer. nat. habil. Dr. h. c. Alexander Schill  
Betreuer: Dipl.-Inf. Josef Spillner



# Selbständigkeitserklärung

Ich versichere, dass ich diese Diplomarbeit selbständig verfasst und nur die angegebenen Quellen, Hilfsmittel und sonstigen Informationsquellen verwendet habe.

Jiakui Li, Dresden, 30. April 2010

.....

*(Unterschrift des Kandidaten)*



# Danksagung

Diese Diplomarbeit entstand im Rahmen des *Dynvocation* Projekts am Lehrstuhl für Rechnernetze des Instituts für Systemarchitektur der Fakultät Informatik der Technischen Universität Dresden unter der Leitung von Herrn Prof. Dr. rer. nat. habil. Dr. h. c. Alexander Schill.

An dieser Stelle möchte ich mich zunächst bei meinem Betreuer Herrn Dipl.-Inf. Josef Spillner bedanken, der mich während meiner Diplomarbeit betreut und umfangreich unterstützt sowie in zahlreichen Treffen und Diskussionen wissenschaftlich in die eingeschlagene Richtung gelenkt hat.

Außerdem möchte ich mich herzlich bei meinen Eltern bedanken, die nicht nur mein Studium zum größten Teil finanziert, sondern auch moralisch sehr unterstützt haben.

Besonders meiner Frau Yi Feng danke ich von ganzem Herzen für ihre unermüdliche Unterstützung, ihre Liebe und Motivation.

Für alles andere danke ich meinen Freunden im In- und Ausland, die mich unterstützt, mir beigestanden haben.

Jiakui Li



# Abkürzungsverzeichnis

<b>API</b>	Application Programming Interface
<b>CSS</b>	Cascading Style Sheets
<b>CTA</b>	Conditional Type Alternatives
<b>DOM</b>	Document Object Model
<b>DTD</b>	Document Type Definition
<b>HTTP</b>	Hypertext Transfer Protocol
<b>IDL</b>	Interface Definition Language
<b>IEEE</b>	Institute of Electrical and Electronics Engineers
<b>INF</b>	Infinity
<b>+INF</b>	Positive Infinity
<b>-INF</b>	Negative Infinity
<b>Infoset</b>	XML Information Set
<b>ms</b>	Millisecond
<b>MVC</b>	Model–View–Controller
<b>NaN</b>	Not a Number
<b>OMG</b>	Object Management Group
<b>OWL</b>	Web Ontology Language
<b>RDF</b>	Resource Description Framework
<b>RegExpInstantiator</b>	Regular Expressions Instances and XML Schema
<b>RELAX</b>	Regular Language description for XML
<b>RELAX NG</b>	Regular Language Description for XML New Generation
<b>RFC</b>	Request for Comments
<b>SAWSDL</b>	Semantic Annotations for WSDL and XML Schema
<b>SAX</b>	Simple API for XML
<b>Saxon EE</b>	Saxon Enterprise Edition
<b>SGML</b>	Standard Generalized Markup Language
<b>SOAP</b>	Simple Object Access Protocol
<b>SVN</b>	Apache Subversion
<b>TREX</b>	Tree Regular Expressions for XML
<b>URI</b>	Uniform Resource Identifier
<b>URL</b>	Uniform Resource Locator
<b>W3C</b>	World Wide Web Consortium
<b>WSDL</b>	Web Services Description Language
<b>Xerces-J</b>	Apache Xerces2 Java
<b>XHTML</b>	Extensible Hypertext Markup Language
<b>XLink</b>	XML Linking Language
<b>XML</b>	Extensible Markup Language
<b>XML4J</b>	IBM's XML Parser for Java
<b>XPath</b>	XML Path Language

<b>XPointer</b>	XML Pointer Language
<b>XSD</b>	XML Schema Definition Language
<b>XSD4J</b>	XML Schema library for Java
<b>XSD4JTest</b>	XML Schema library for Java Test Tool
<b>XSL</b>	Extensible Stylesheet Language
<b>XSLFO</b>	Extensible Stylesheet Language – Formatting Objects
<b>XSLT</b>	XSL Transformation
<b>XSWG</b>	XML Schema Working Group



# Inhaltsverzeichnis

<b>Kapitel 1</b>	<b>Einleitung</b>	<b>1</b>
1.1	Die Ziele dieser Arbeit	1
1.2	Der Aufbau dieser Arbeit	1
<b>Kapitel 2</b>	<b>Theoretische Grundlagen</b>	<b>5</b>
2.1	XML	5
2.1.1	Die Regeln der Wohlgeformtheit	6
2.1.2	Wohlgeformtes XML- Dokument	6
2.2	Namensräume (Namespaces)	7
2.3	XPath	7
2.4	DTD	8
2.5	XML-Schema	8
2.6	Vergleich des XML-Schemas mit DTD	9
2.7	Implementierung der XML-Bibliotheken Xerces-J und Saxon EE	9
2.7.1	Xerces-J	10
2.7.2	Saxon EE	10
2.8	Zusammenfassung	10
<b>Kapitel 3</b>	<b>Inhaltsmodelle mit dem XML-Schema</b>	<b>11</b>
3.1	Erklärungen der Terminologie	11
3.1.1	Schema	11
3.1.2	Beziehung zwischen Instanz und Schema	11
3.1.3	Deklaration und Typdefinition	12
3.2	Die Komponenten eines XML-Schemas	12
3.3	Struktur des XML-Schema Dokuments	13
3.3.1	Deklaration von Elementen	13
3.3.2	Deklaration von Attributen	15
3.3.3	Globale und lokale Deklarationen	15
3.3.4	Kompositoren	16
3.3.5	Partikel	18
3.3.6	Mehrfache XML-Dokumente und -Namensräume	19
3.3.7	Identitätseinschränkungen	21
3.3.8	Definition von Element- und Attributgruppen	21
3.3.9	Ableitung komplexer Datentypen	22
3.3.10	Kommentare	23
3.4	Datentypen	24
3.4.1	Komplexe Datentypen	25
3.4.2	Vordefinierte einfache Datentypen	25

3.4.3	Primitive Datentypen . . . . .	25
3.4.4	Abgeleitete einfache Datentypen . . . . .	26
3.4.5	Atomare Datentypen, Listen- und Vereinigungs-Datentypen . . . . .	27
3.4.6	Einschränkende Facetten für Datentypen . . . . .	29
3.4.7	Liste der einschränkenden Facetten. . . . .	29
3.5	Vergleich mit anderen Schemasprachen . . . . .	31
3.5.1	RELAX NG . . . . .	31
3.5.2	Schematron . . . . .	31
3.6	Zusammenfassung. . . . .	32

## **Kapitel 4 Analyse der Architektur und Umsetzung von XSD4J 33**

4.1	Die XSD4J Bibliothek. . . . .	33
4.1.1	XML Parser. . . . .	33
4.1.2	DOM . . . . .	34
4.1.3	XSDJ mit dem DOM . . . . .	34
4.2	Architektur von XSD4J. . . . .	35
4.2.1	APIs in der XSD4J Bibliothek . . . . .	36
4.2.2	Der Implementierungsablauf der Klasse XSDParser. . . . .	40
4.2.3	Der Implementierungsablauf der Klasse XSDDumper. . . . .	44
4.3	Zusammenfassung. . . . .	46

## **Kapitel 5 Arbeit mit XSD 1.1 47**

5.1	Einführung der XSD 1.1. . . . .	47
5.2	Die neuen Strukturen. . . . .	48
5.2.1	Namensräume für XSD 1.1. . . . .	50
5.2.2	Schema-Namensraum-Version . . . . .	50
5.2.3	Deklaration von Assertions . . . . .	51
5.2.4	Element <xsd:assert>. . . . .	52
5.2.5	Bedingter Typ <xsd:alternative>. . . . .	52
5.2.6	Vergleich von <xsd:assert > und <xsd:alternative>. . . . .	55
5.2.7	Deklaration von defaultAttributes (Schema-wide-Attributes). . . . .	56
5.2.8	<xsd:openContent> und <xsd:defaultOpenContent>. . . . .	57
5.2.9	Ungeordneter Inhalt für das Element <xsd:all>. . . . .	58
5.3	Die neuen Facetten. . . . .	62
5.3.1	Die Facette <xsd:assertion>. . . . .	62
5.3.2	Die Facette <xsd:explicitTimezone> . . . . .	62
5.3.3	Die Facetten <xsd:minScale> und <xsd:maxScale> . . . . .	63
5.4	Die neuen Datentypen . . . . .	64
5.4.1	Der Datentyp <xsd:anyAtomicType>. . . . .	67
5.4.2	Der Datentyp <xsd:precisionDecimal>. . . . .	68
5.4.3	Der Datentyp <xsd:dateTimeStamp>. . . . .	69

5.4.4	Der Datentyp <xsd:yearMonthDuration> . . . . .	70
5.4.5	Der Datentyp <xsd:dayTimeDuration> . . . . .	72
5.5	Zusammenfassung . . . . .	73

## **Kapitel 6 Die Testfälle von XSD 75**

6.1	Konzept der Testfälle . . . . .	75
6.2	Implementierung der XSD4JTest . . . . .	76
6.2.1	Das Model-View-Controller-Entwurfs-Muster(MVC) . . . . .	76
6.2.2	Architektur von XSD4JTest . . . . .	77
6.3	Definitionen der Struktur des Dokuments( Testfälle) . . . . .	78
6.4	Einzelner Testfall und Mehrere Testfälle . . . . .	79
6.4.1	Testfälle für XSD 1.1 (Implementierung) . . . . .	79
6.4.2	Testfälle für XSD 1.0 und XSD 1.1 (W3C) . . . . .	80
6.4.3	Testfälle für XSD 1.1 (Apache Xerces2 Java) . . . . .	82
6.4.4	Testfälle für XSD 1.1 (Saxon EE) . . . . .	83
6.5	Vorstellung der APIs, Servlets und JSPs. . . . .	85
6.6	Zusammenfassung. . . . .	88

## **Kapitel 7 Erweiterung der XSD4J Bibliothek für SAWSDL 89**

7.1	Überblick von SAWSDL . . . . .	89
7.2	Die Liste der Merkmale von SAWSDL im XSD . . . . .	90
7.3	XSD4J-Erweiterung für SAWSDL. . . . .	91
7.3.1	modelReference . . . . .	91
7.3.2	loweringSchemaMapping und liftingSchemaMapping . . . . .	91
7.4	Zusammenfassung . . . . .	92

## **Kapitel 8 Zusammenfassung und Ausblick 93**

8.1	Zusammenfassung. . . . .	93
8.2	Ausblick . . . . .	95

## **Literaturverzeichnis . . . . . 97**

## **Anhang . . . . . 105**

<b>A.</b>	<b>Dokumentation von XSD . . . . .</b>	<b>105</b>
<b>B.</b>	<b>Zusätzliche Tabellen. . . . .</b>	<b>107</b>
<b>C.</b>	<b>Screenshots . . . . .</b>	<b>111</b>
<b>D.</b>	<b>Index der Abbildungen, Tabellen und Listings . . . . .</b>	<b>117</b>



# Kapitel 1

## Einleitung

Die vorliegende Arbeit entsteht aus der Erweiterung der existierten XSD4J Bibliothek, die zu einem Teil des Projekts Dynvocation<sup>1</sup> gehört. Die XSD4J Bibliothek SVN-Version (Standard: 1426) hat die hauptsächliche Unterstützung für die XML-Schema 1.0 Spezifikation realisiert. Aufgrund der Aktualisierung der neuen Vision von XSD 1.1 bedarf diese XSD4J Bibliothek weiterer Vervollständigung, damit die neuen Funktionen unterstützt werden können.

### 1.1 Die Ziele dieser Arbeit

Wie bereits in der Aufgabestellung erläutert, beschäftigt sich diese Arbeit mit der Vervollständigung der XML-Schema-Bibliothek XSD4J. In den folgenden Auflistungen sind die Aufgabenstellungen dieser Arbeit aufgeführt:

- Einführung der für diese Arbeit wichtigen Grundlagen in XSD 1.0 und die relativen XML-Familien-Spezifikationen sowie Erläuterung der relationalen Umgebungssoftware, besonders der integrierbaren Benutzung von XSD4J.
- Analyse der Parser von XSD4J, der inneren Struktur und der realisierten Funktionen und Eigenschaften.
- Realisierung der überprüfbareren Testfälle und auch der Überprüfung im einzelnen Testfall.
- Entwicklung und Implementierung der Testfällen für die neue Version XSD 1.1, um die neuen Merkmale zu unterstützen
- Die automatische Unterstützung im *Bootstrapped* Schemata<sup>2</sup> von XSD 1.0 und XSD 1.1
- Die erweiternde Entwicklung der transparenten Unterstützung der neuen Version XML-Schema 1.1 sowie der auf XML-Schema basierenden Erweiterungen wie SAWSDL für die semantische Annotationen der Schemakonstrukte.
- Überblick der realisierten Funktionen in der XSD4J Bibliothek und die Zusammenfassung dieser Arbeit.

### 1.2 Der Aufbau dieser Arbeit

Im vorigen Abschnitt wurden bereits die Ziele dieser Arbeit vorgestellt. Um den Aufbau der Arbeit zu verdeutlichen, werden die einzelnen Kapitel kurz beschrieben.

---

<sup>1</sup><http://dynvocation.selfip.net/xsd4j/>

<sup>2</sup>Die XSD4J Bibliothek kann automatisch zwischen `BootstrappedSchema10` und `BootstrappedSchema11` wählen. Diese beiden Klassen realisieren die Unterstützungen der XSD 1.0 und XSD 1.1 Spezifikationen.

**Kapitel 2** Dieses Kapitel beschreibt einerseits, welche Mitglieder es in der XML-Familie gibt, die Bezug auf das XML-Schema nehmen, andererseits werden Saxon<sup>3</sup> und Xerces-J<sup>4</sup> vorgestellt. Diese unterstützen auch die XSD 1.1 Spezifikation in der aktuellen Version.

**Kapitel 3** Dieses Kapitel enthält die Inhaltsmodelle mit dem XML-Schema. Dabei handelt es sich um den Aufbau eines XML-Schemas, die Strukturen und die Datentypen der XML-Schema-Spezifikation sowie die Erläuterung der jeweiligen Komponenten. Des Weiteren werden in diesem Kapitel die andere Schemasprachen RELAX NG und Schematron kurz vorgestellt und mit dem XML-Schema hinsichtlich der Vorteile und Nachteile verglichen.

**Kapitel 4** In diesem Kapitel wird zunächst die XSD4J Bibliothek näher vorgestellt. Dann beschreibt es hauptsächlich die interne Architektur, die eingehende Analyse von Parser und die Funktionen von der XSD4J Bibliothek. Anschließend liefert dieses Kapitel die Erläuterung einer intensiven Dokumentation der APIs der XSD4J Bibliothek. Diese APIs beschreiben die Methoden XSDParser und XSDDumper. Im Anschluss zeigt das Kapitel anhand eines Beispiels, der Ergänzung des Elements `<xsd:annotation>` von XSD 1.1, wie die Implementierung für die Vervollständigung der XSD4J Bibliothek programmiert ist.

**Kapitel 5** XSD 1.1 ist die Obermenge von XSD 1.0. Es hat mehrere leistungsstarke Funktionalitäten. In diesem Kapitel wird die aktuelle neue Version XSD 1.1 erklärt. Es werden die neuen Strukturen und Datentypen der XSD 1.1 Spezifikation näher betrachtet. Die neuen Strukturen sind dabei z.B. das Schema Version Namensraum, Assertions `<xsd:assert>`, konditionale Typen `<xsd:alternative>`, Schema-wide-Attributes, *Open Content* und ungeordnete Inhalte für das Element `<xsd:all>`. Die neuen Datentypen sind dabei z.B. `<xsd:anyAtomicType>`, `<xsd:precisionDecimal>`, `<xsd:dateTimeStamp>`, `<xsd:yearMonthDuration>` und `<xsd:dayTimeDuration>`. Die vier neuen Facetten sind `<xsd:maxScale>`, `<xsd:minScale>`, `<xsd:assertion>` und `<xsd:explicitTimezone>`. Dieses Kapitel wird diese neuen Merkmale ausführlich anhand des Beispiels beschreiben.

**Kapitel 6** In diesem Kapitel werden die Testfälle überprüft, die jeden einzelnen üblichen Anwendungsfall und die Beispielszenarien sowie die mehrere Anwendungsfälle beinhalten. Das Test-Werkzeug XSD4JTest wird entwickelt, um die Testfälle zu überprüfen. Am Ende wird das Ergebnis der Testfälle in Grafiken und Tabellen überführt, um eine Analyse der aktuellen Version von XSD4J zu ermöglichen.

**Kapitel 7** In diesem Kapitel wird zunächst die Anwendung der SAWSDL<sup>5</sup> Spezifikation in der XSD4J Bibliothek erläutert, dann dient es vor allem dem Zweck, die Erweiterung für die Unterstützung von SAWSDL in der XSD4J Bibliothek zu zeigen.

---

<sup>3</sup><http://saxon.sourceforge.net/>

<sup>4</sup><http://xerces.apache.org/xerces-j/>

<sup>5</sup>Semantic Annotations for WSDL and XML-Schema: <http://www.w3.org/TR/sawSDL/>

**Kapitel 8** Dieses Kapitel schließt mit einer Zusammenfassung dieser Arbeit und formuliert Empfehlungen für weitere Forschungs- und Entwicklungsansätze.





# Kapitel 2

## Theoretische Grundlagen

Das folgende Kapitel beschreibt zunächst die für das Verständnis des Themas notwendigen Grundlagen. Dabei wird auf XML und dessen Familie mit den Begriffen Namensraum, XPath und DTD/XML-Schema, sowie auf die Implementierung der XML-Bibliotheken Xerces-J [54] und Saxon EE [56] näher eingegangen.

### 2.1 XML

XML<sup>1</sup> ist die Abkürzung von Extensible Markup Language und beschreibt eine Klasse von Datenobjekten, die sogenannte XML-Dokumente sind. Der Entwickler kann diese Sprache auf Basis der eigenen Anforderungen definieren und manipulieren. XML ist auch ein universales Datenaustauschformat der Strukturen und die grundsätzliche Syntax der Auszeichnungssprachen[1]. XML wird als Nachfolger der HTML gehandelt und kann auf die Separation zwischen den Daten und der Repräsentation eingehen. Gleichzeitig verfügt XML über Wohlgeformtheits- und Validierungsmechanismen sowie einen öffentlichen Standard [67].

XML besitzt eine Reihe von verbundenen Technologien, die die Validierung, Repräsentation, Transformation, Links von Dateien und DOM (Document Object Model)<sup>2</sup> enthalten.

Die Abbildung 2.1 zeigt zunächst einen kurzen Überblick mit Hinweisen auf den Status. Die Referenzen[2] sind im Einzelnen:

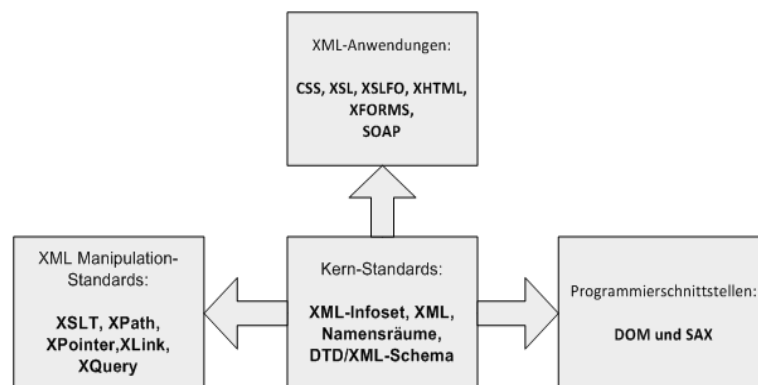


Abbildung 2.1: Überblick über die Sprachfamilie XML [3]

<sup>1</sup>Den schnellsten Zugang zum aktuellen Stand aller Standardisierungsprojekte des W3C finden bietet die Adresse: <http://www.w3.org/standards/xml>

<sup>2</sup><http://www.w3.org/DOM/>

- Kern-Standards: XML InfoSet [81], XML Namensräume und DTD/XML-Schema.
- Manipulation-Standards: XSLT, XPath, XPointer, XLink und XQuery.
- Programmierschnittstellen: DOM und SAX.
- XML-Anwendungen: CSS, XSL, XSLFO, XHTML, XFORMS und SOAP [4].

Den Kern der Sprachfamilie bildet die XML-Spezifikation, ihre Erweiterungen sind XML-Namensräume und die Sprache zur Definition von Inhaltsmodellen XML-Schema. Die theoretische Basis dieser W3C Empfehlungen, die direkt den Inhalt von XML-Dokumenten betreffen, wurde noch einmal separat als XML Information Set (InfoSet) formuliert, um einen konsistenten Satz von Definitionen für alle Spezifikationen rund um XML zur Verfügung zu stellen.

DOM definiert die Programmierschnittstellen, mit denen auf alle Informationen aus XML-Dokumenten zugegriffen werden kann und die gewonnenen Informationen bearbeitet werden können. Die XSD4J Bibliothek verwendet DOM zur Bearbeitung des XSD-Dokuments.

Um Plattform- und Programmiersprachenunabhängigkeit zu gewährleisten, wurde zur Beschreibung der Schnittstellen die OMG Interface Definition Language [64](OMG IDL) benutzt.

### 2.1.1 Die Regeln der Wohlgeformtheit

Jede Sprache hat Regeln. In menschlichen Sprachen nehmen diese Regeln viele Formen an: Wörter haben eine bestimmte korrekte Aussprache, manchmal auch mehrere und sie können auf verschiedene Weise kombiniert werden, um gültige Sätze zu bilden (Grammatik) [5]. Wenn ein Dokument die in den XML-Spezifikationen beschriebenen Regeln erfüllt, bezeichnet man es als wohlgeformt (*well-formed*). Die Wohlgeformtheit eines Dokumentes (well-formed documents) besagt nicht nur im Grunde, dass sich das Dokument vollständig an die offiziellen Regeln des W3C zur Erstellung von XML-Dokumenten hält, sondern die Wohlgeformtheit ist auch die Voraussetzung dafür, dass eine XML-Datei verarbeitet werden kann.

Das wohlgeformte Dokument besitzt einige Vorteile [52]. z.B. :

- Es kann das XML-Dokument ohne DTD bauen, um den besseren Datenaustausch zu verwenden
- Der XML-Prozessor kann kleiner sein und schneller laufen.

### 2.1.2 Wohlgeformtes XML-Dokument

Die Regeln für wohlgeformte XML-Dokumente folgen hauptsächlich den folgenden Bestimmungen nach der W3C-Empfehlung:

Ein XML-Dokument [6]:

- besitzt nur ein Wurzelement.
- muss aus mindestens einem Element bestehen.
- muss entweder ein Start- und Endtag haben (`<element>...</element>`) oder als alleinstehender Tag ausgezeichnet sein (`<element/>`)
- muss seine Start- und Endtags korrekt verschachteln. Das heißt, dass der

Endtag in derselben Ebene wie der Starttag liegen muss. (D.h., eine Folge der Art `<a> <b> </a> </b>` kommt nicht vor. Dies kann der Entwickler durch Einrückungen entsprechend der Ebenen verdeutlichen, am Aufbau des XML-Dokuments ändert das jedoch nichts)

- darf seine Tag- und Attributnamen keine Leerzeichen enthalten (entgegen einer weitverbreiteten Meinung sind auch Großbuchstaben und sogar einige Sonderzeichen wie Umlaute erlaubt, der Tag-Name ist hierbei *nicht* wie bei SGML *case-insensitive*)
- sein Wert eines Attributs wird immer in Anführungszeichen geschrieben (*attr='value' oder attr="value"*)

## 2.2 Namensräume (Namespaces)

Die Namensräume-Spezifikation definiert einen Mechanismus, mit dem Element- und Attributnamen eines Vokabulars eindeutig zu identifizieren sind. Sie basieren auf der XML-Syntax und werden mit URIs kombiniert.

Ein Identifier kann von mehreren Namensräumen definiert sein und ein neuer Namensraum kann wiederum beliebige Identifier definieren. In XML-Dokumenten werden Namensräume als Element- und Attributnamen verwendet [7].

Bevor die Namensräume im XML-Dokument verwendet werden, müssen die Namensräume deklariert werden. Die Namensraumdeklaration wird normalerweise verwendet, um einen Namensraum-URI einem spezifischen Präfix zuzuordnen. Der Bereich der Namensräume deklariert sein Element und alle untergeordneten Elemente. Die Default-Namensräume sind eine Ausnahme, weil sie ohne Angabe eines Präfixes deklariert werden [59].

Das in Listing 2.2 abgedruckte Element `<xsd:schema>` enthält die Darstellungen der verschiedenen Namensräume im XML-Schema.

Listing 2.2: Die Darstellungen der verschiedenen Namensräume im XML-Schema

```
<xsd:schema xmlns:xsd="http://www.w3.org/2001/XMLSchema"
xmlns:xhtml="http://www.w3.org/1999/xhtml"
xmlns="http://www.example.com"
targetNamespace="http://www.example.com">
...
</xsd:schema>
```

## 2.3 XPath

XPath ist die XML Path Language. Sie kann die Position innerhalb des XML-Dokuments festlegen und basiert auf dem XML-Baum. Sie kann die Knoten in den strukturierten Daten des XML-Dokuments finden und wird hauptsächlich in Elementen und Attributen verwendet[8]. XPath wird nicht nur von XSL-Transformations (XSLT) verwendet, sondern auch von anderen XML-Sprachen wie XML-Schema. Sie ist eine sehr kompakte Sprache

mit einer Syntax, die Pfadausdrücke widerspiegelt, wie sie aus Dateisystemen bereits bekannt sind. Diese Pfadausdrücke nennt man XPath-Ausdrücke. Sie sind jedoch generalisiert und damit sehr viel mächtiger als die vergleichsweise einfachen Pfadausdrücke für Dateisysteme. Aufgrund ihrer Verwendung in vielen verschiedenen XML-Sprachen ist XPath eine der wichtigsten XML Sprachkomponenten [9].

Eine Teilmenge von XPath wird im XML-Schema für die Bildung von den Werten für spezielle Auswahlattribute in den Elementen `<xsd:key>`, `<xsd:unique>`, `<xsd:keyref>` benutzt.

### 2.4 DTD

Eine Document Type Definition (DTD) definiert eine bestimmte Klasse von Dokumenten, die alle von dem gleichen Typ sind, indem sie verbindlich das Vokabular und die Grammatik für die Auszeichnungssprache festlegt, die bei der Erstellung des Dokuments verwendet werden soll und darf.

Die DTD entspricht damit weitgehend den Grammatikregeln einer Programmiersprache und jedes XML-Dokument, das einer solchen DTD genügt, einem syntaktisch korrekten Programm [10].

Ein XML-Dokument, das nicht nur wohlgeformt ist, sondern auch der DTD genügt, heißt gültiges Dokument (valid document), wenn es sich an die dort aufgestellten Strukturierungsvorschriften hält. Das heißt, dass diese Eigenschaft des wohlgeformten Dokuments unabhängig von der Existenz einer DTD ist.

### 2.5 XML-Schema

XML-Schema ist eine Alternative gegenüber der Verwendung von DTDs zur Beschreibung von XML-Auszeichnungssprachen. Die Datentypen können in einem XML-Schema ähnlich wie in einer Programmiersprache definiert werden. Zusätzlich gibt es u.a. die Möglichkeit, den Inhalt von Elementen und Attributen auf Zahlenbereichen zu beschränken oder zulässige Texte durch reguläre Ausdrücke zu definieren. Ein XML-Schema ist selbst ein XML-Dokument und wird meist in einer Datei mit der Endung `.xsd` gespeichert [46].

Ein XML-Schema besteht aus die verschiedenen Komponenten. Diese können verwendet werden, um die Gültigkeit der wohlgeformten Elemente zu prüfen. Auf der anderen Seite können sie die Erweiterung solcher Informationseinheiten und ihre Nachkommen spezifizieren. Diese Erweiterung stellt die Informationen, die möglicherweise implizit im originalen Dokument existieren, z.B. normalisierte und / oder Default-Werte für Attribute und Elemente sowie Typen von Element-und Attribut-Informationseinheiten, zur Verfügung.

Die Schemagültigkeitsprüfung hat zwei Aspekte [11]:

1. Für das Dokument wird überprüft, d.h., ob seine Struktur der im Schema festgelegten Struktur entspricht.
2. Die Informationen aus dem Schema werden der Informationmenge des Instanzdokumentes hinzugefügt, damit sie für die Weiterverarbeitung zur Verfügung stehen. Zu dieser Informationsmenge gehören z.B. Default-Werte für

nicht angegebene Attribute oder Typinformationen usw. aber auch normale Datentypen von dem XML-Schema.

Mit einem XML-Schema komplexere Zusammenhänge als mit einer DTD können beschreiben und DTDs werden irgendwann vollständig von XML-Schema abgelöst. Um komplexe Datentypen zu beschreiben, kann man Schemata benutzen. Die XML-Schema Spezifikation ist zurzeit vom W3C als eine Schemasprache definiert und bietet eine sehr große Ausdrucksfähigkeit.

## 2.6 Vergleich des XML-Schemas mit DTD

Obwohl die DTD erfolgreich im Bereich der SGML und ein etablierter Mechanismus für die Beschreibung von den strukturierten Informationen bei HTML-Entwicklern ist, gibt es erhebliche Einschränkungen im Vergleich mit dem XML-Schema.

Während DTDs recht einfach strukturiert sind und nur grobe Einschränkungen für das Instanzdokument vorgeben können, erlaubt das XML-Schema die Verwendung der üblichen Datentypen, genaue Angaben von Kardinalitäten und insgesamt eine viel differenziertere Beschreibung der gewünschten Strukturen [12].

Die DTD verwendet keine XML-Syntax, deshalb bietet es nur eingeschränkte Unterstützung der Datentypen und der Namensräume. Aber XML-Schema ist selbst eine XML-Auszeichnungssprache, in der verschiedene Elementtypen zur Definition der Struktur von XML-Instanzen zur Verfügung stehen, sowie diverse Attribute dieser Element Typen, mit denen weitere einschränkende Angaben gemacht werden können. Die Tabelle 2.1 zeigt die Nachteile von DTDs und die Vorteile des XML-Schemas [13]:

Nachteile von DTDs	Vorteile des XML-Schemas
<ol style="list-style-type: none"> <li>1. DTD ist selbst kein XML-Dokument.</li> <li>2. DTD ist Dokument-orientierte Ausdruck.</li> <li>3. verschiedenen Datentypen nur für Attribute, nicht für Elemente.</li> <li>4. keine Unterstützung für Namensräume.</li> </ol>	<ol style="list-style-type: none"> <li>1. XML-Schema ist selbst XML-Dokument</li> <li>2. stärkere und schwächere Constraints als DTDs</li> <li>3. verschiedene Datentypen jetzt auch für Elemente (strings, integers, Boolesche Werte usw. )</li> <li>4. vordefinierte Typen und benutzerdefinierte Typen</li> <li>5. objekt-orientierte Konzepte: strukturelle Vererbungsmechanismen zwischen Typen(Typenhierarchie)</li> <li>6. modulare Definition</li> <li>7. Unterstützung für Namensräume</li> </ol>

Tabelle 2.1: Vergleich des XML-Schemas mit DTD

## 2.7 Implementierung der XML-Bibliotheken Xerces-J und Saxon EE

Um die Bibliothek XSD zu überprüfen, werden die Testfälle aus Xerces-J und Saxon EE genutzt. Die verschiedenen Testfälle haben schon teilweise die Spezifikation XSD 1.1

unterstützt. Xerces-J und Saxon EE werden im Folgenden erläutert.

### 2.7.1 Xerces-J

*Apache Xerces2 Java* (Xerces-J) ist ein Prozessor für den Parser, Validierung, Serialisierung und Verarbeitung von XML, geschrieben in Java [14]. Der Xerces-J Parser gehört zur Apache-Gruppe, sie entstand aus dem IBM-Parser XML4J. Die aktuelle Version von Xerces-J ist 2.10.0 (SVN Version: 926149). Diese Version bietet natürlich validierende und nichtvalidierende SAX- und DOM-Parser. Außerdem ist Xerces-J der einzige Parser, der schon XML-Schema unterstützt [43].

Die Eigenschaften dieser aktuellen Unterstützung [15] von XSD 1.1 enthalten *assertions*, meist Anteile von *conditional type assignment* (CTA), Implementierung des Datentyps `<xsd:error>`, *open content*, den neuen Datentyp `<xsd:precisionDecimal>` und die neuen Regeln für die Einschränkung des komplexen Typs.

Die in dieser Arbeit beschriebenen Testfälle verwenden XSD 1.1 der aktuellen Version.

### 2.7.2 Saxon EE

Saxon EE unterstützt XSLT 2.0, XPath 2.0, XQuery 1.0 und XML-Schema 1.0 und auch den höheren Entwurf der Spezifikationen XQuery Update 1.0, XQuery 1.1, XSD 1.1 und XSLT 2.1<sup>3</sup>. *Saxon EE 9.2.0.2 Java* hat die neuen Merkmale bei dem XSD 1.1 unterstützt. Die Eigenschaften dieser Unterstützung von XSD 1.1 enthalten *assertions*, *conditional type assignment* (CTA), *open content* sowie die neuen Regeln für die Einschränkung des komplexen Typs und die Partikel *unique*. Diese Merkmale sind nach der Entwicklung der W3C-Spezifikation verändert.

Saxon EE enthält ein paar Beispiele für die Version XSD 1.1. In dieser Arbeit werden diese Beispiele angewendet, um die XSD4J Bibliothek zu überprüfen. Außerdem bietet XFRONT<sup>4</sup> auch die Verweise der neuen Merkmale von XSD 1.1 an, die von Saxon EE zu überprüfen sind [16].

## 2.8 Zusammenfassung

In diesem Kapitel wurden die notwendigen theoretischen Grundlagen der Technologie XML eingeführt. Thematisiert wurden die Regeln der Wohlgeformtheit, das wohlgeformte XML-Dokument, Namensräume, XPath, DTD und XML-Schema. Es folgte den Vergleich von XML-Schema und DTD, um die Vorteile des XML-Schemas zu zeigen. Dieses Kapitel zeigt am Ende, welche Testfälle in der Implementierung der XSD4J Bibliothek angewendet wurden, z. B. Xerces-J und Saxon EE.

In dem folgenden Kapitel wird es darum gehen, die Inhaltsmodelle mit dem XML-Schema zu beschreiben, weil die Hauptaufgabe der XSD4J Bibliothek die Umsetzung des XML-Schemas ist.

---

<sup>3</sup><http://saxon.sourceforge.net/#F9.2EE>

<sup>4</sup><http://www.xfront.com/xml-schema-1-1>

## Kapitel 3

### Inhaltsmodelle mit dem XML-Schema

Die XML Schema Description Language (XSD) wurde von der XML Schema Working Group empfohlen. Sie kann mit XML, XPath, den Namensräumen, und anderen XML-basierten Spezifikationen zusammenarbeiten. In der Spezifikation wurde XSD in drei Teilen geteilt. Unter Part:0 [17] ist eine zusammenfassende Einführung zu finden. Part 1: *Structures* [11] behandelt die Definitionen von Inhaltsmodellen für Element- und Attributdeklarationen und Strukturen im XSD-Dokument. Part 2: *Datatypes* [19] definiert die einfachen Datentypen, die in Part 1 verwendet werden.

Nachfolgend werden die Bestandteile der Schemasprache XSD an einigen Beispielen eingeführt: die Strukturen und Datentypen.

#### 3.1 Erklärungen der Terminologie

XML-Schema basiert auf einer Zahl von Begriffen. Um sie gut zu verstehen, wird dieser Abschnitt zunächst die Bedeutung einiger Begriffe erklären.

##### 3.1.1 Schema

Üblicherweise ist das Schema selbst in einer formalen Sprache definiert, so dass sich Daten automatisch darauf überprüfen lassen, ob sie dem Schema entsprechen [20]. XML-Schema für XML ist ein bekanntes Beispiel für eine solche Beschreibungssprache. Alle durch XSD definierten Elemente, d.h. alle Primitive zur Definition eines eigenen Schemas, befinden sich im Namensraum <http://www.w3.org/2001/XMLSchema>, der üblicherweise an das Präfix *xsd* gebunden wird. Elemente und Attribute aus XML-Schema, die in Instanzdokumenten verwendet werden, sind im Namensraum <http://www.w3.org/2001/XMLSchema-instance> (übliches Präfix *xi*) organisiert.

Das Schema besteht aus mehreren verschiedenen Typen, z.B. Element, Attribut, einfache Typen, komplexe Typen, Notation, Kommentare, Elementgruppen, Attributgruppen und Facetten. Im XML-Schema-Dokument muss das Schema mit dem Wurzelement *xsd:schema* verwendet werden.

##### 3.1.2 Beziehung zwischen Instanz und Schema

Das Ziel eines Schemas ist es, dass eine Klasse der XML-Dokumente definiert ist. Deshalb wird häufig der Begriff „Instanzdokument“ oder kurz „Instanz“ verwendet, um ein Dokument zu beschreiben, das einem bestimmten Schema entspricht [80]. Der Übergang des überprüften Instanzdokuments, das einem Schema oder mehreren Schemata entspricht, ist die so genannte „Validierung“ (schema validation).

Die Abbildung 3.1 stellt die getroffenen Aussagen und Validierungsbeziehungen über XML-Schema nochmals grafisch zusammen.

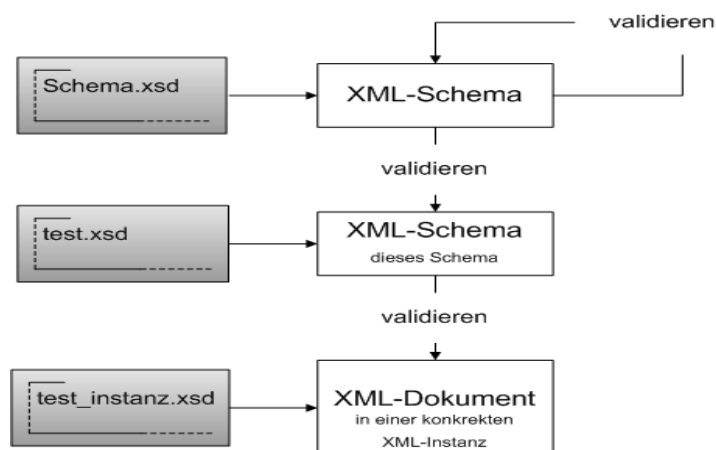


Abbildung 3.1: Die Beziehung zwischen Instanz und Schema

### 3.1.3 Deklaration und Typdefinition

In der XML-Schema-Spezifikation gibt es zwei Begriffe: „Deklaration“ und „Typdefinition“. Es gibt drei Arten von Deklarations-Komponenten [21]: Element, Attribut und Notation. Typdefinitionen bilden eine Hierarchie mit einer einzigen Wurzel. Die nachfolgenden Unterabschnitte beschreiben zunächst die Merkmale dieser Hierarchie; anschließend werden Definitionen des einfachen und komplexen DatenTyps eingeführt. Die folgende Deklaration *geburtsdatum* definiert ein XML-Element des Typs `<xsd:date>` zur Darstellung eines Datums:

```
<element name="geburtsdatum" type="xsd:date"/>
```

## 3.2 Die Komponenten eines XML-Schemas

Die W3C XML-Schema-Spezifikation empfiehlt drei Gruppen von Komponenten, aus denen ein Schema zusammengefügt werden kann. Die erste Gruppe umfasst die primären Komponenten: einfache Typdefinitionen, komplexe Typdefinitionen, Elementdeklarationen und Attributdeklarationen. Während für die Deklaration von Elementen und Attributen die Vergabe von entsprechenden Namen notwendig ist, können Typdefinitionen auch ohne Namen eingesetzt werden.

In die zweite Gruppe werden die folgenden Komponenten eingestuft: Definitionen von Attributgruppen, Eindeutigkeitsbeschränkungen und Schlüsselreferenzen, Modellgruppendifinitionen und Deklarationen von Anmerkungen. Bei diesen Komponenten ist die Zuordnung von Namen zwingend.

Die dritte Gruppe besteht aus Hilfskomponenten, die immer nur als Teile von anderen Komponenten auftreten können: Anmerkungen, Modellgruppen, Partikel, Wildcards und Festlegungen über die Verwendung von Attributen [22]. Abbildung 3.2 zeigt die Anwendung der Komponenten im XSD-Dokument.



schema	Gruppe1	attribute	simpleType	restriction	Facetten	
		element	complexType	complexContent	extension	Kompositoren
	Gruppe2	attributeGroup	Attribut1			
			Attribut2			
			Attribut3			
		element	key			
			keyref			
			unique			
		group	sequence	Element1		
				Element2		
				Element3		
		notation				
	Gruppe3	annotation	documentation	any	@minOccurs	1
					@maxOccurs	unbounded
					@namespace	##targetNamespace
			appinfo			
		complexType	sequence	choice	Element1	
					Element2	
					Element3	
			attribute	@name	Attribut-Verwendungen	
				@type	xsd:string	

Abbildung 3.2: Die Komponenten eines XML-Schemas

### 3.3 Struktur des XML-Schema-Dokuments

XML-Schemaelemente werden über ihre Funktionen zu Gruppen zusammengefasst: Elemente, Attribute, Kompositoren, Partikel, mehrfache XML-Dokumente und -Namensräume, Identitätseinschränkungen, benannte Attribute, komplexe Typdefinitionen und einfache Typdefinitionen [23]. Im folgenden Abschnitt wird die Struktur des XML-Schema-Dokuments vorgestellt.

#### 3.3.1 Deklaration von Elementen

Die Deklaration von Elementen geschieht mithilfe des Elements `<xsd:element>`. Im XSD-Dokument gibt es zwei Elementarten: komplexe Elemente und einfache Elemente.

##### I. Komplexe Elemente

In Listing 3.3.1.1 wird eigens für dieses Schema ein komplexer Datentyp entworfen, der dem Element über den Typnamen `booklist` zugewiesen wird.

Die Definition dieses komplexen Datentypes gibt an, wie die verschiedenen Unterelemente von `book` gruppiert werden sollen. Für die Bildung einer solchen Modellgruppe wird der Kompositor `<xsd:sequence>` verwendet, der als Kind des Elements `<xsd:complexType>` eingefügt wird.

Listing 3.3.1.1: Beispiel für komplexe Elemente

```
<xsd:element name="book" type="booklist"/>
<xsd:complexType name="booklist">
  <xsd:sequence>
    <xsd:element name="computer" type="computerType"/>
    <xsd:element name="music" type="musicType"/>
  </xsd:sequence>
</xsd:complexType>
```

Innerhalb der Sequenz werden nacheinander zwei Elemente deklariert. In beiden Fällen wird neben dem Elementnamen wieder der Datentyp angegeben. Sie gehören zu zwei komplexen Datentypen.

## II. Einfache Elemente

Die Definition des komplexen Types *computerType* zeigt, dass auch hier wieder mit einer Sequenz von Elementen gearbeitet wird. Es handelt sich um einfache Datentypen in der Sequenz eingefügter Elemente [70]. Einfache Datentypen enthalten selbst keine Kindelemente und führen auch keine Attribute mit, sie werden von XML-Schema vordefiniert, z.B. *<xsd:string>*. Deshalb wird hier jedes Mal das Präfix *xsd* verwendet. Listing 3.3.1.2 zeigt ein Beispiel für die Anwendung einfacher Elemente.

Listing 3.3.1.2: Beispiel für einfache Elemente

```
<xsd:complexType name="computerType">
  <xsd:sequence>
    <xsd:element name="title" type="xsd:string"/>
    <xsd:element name="author" type="xsd:string"/>
  </xsd:sequence>
</xsd:complexType>
```

Zusätzlich zur Angabe des Datentyps kann eine Elementdeklaration auch eine Angabe zu einem Default-Wert oder auch einen fixen Wert enthalten. Wenn das Element *title* in den Datentyp *computerType* als Default-Wert definiert wird, sind folgende Ausdrücke möglich:

```
<xsd:element name="title" type="xsd:string" fixed="XSD4J Bibliothek"/>
```

oder

```
<xsd:element name="title" type="xsd:string" minOccurs="0" default="XSD4J Bibliothek"/>
```

Quelle: [http://www.w3.org/TR/xmlschema-1/#cElement\\_Declarations](http://www.w3.org/TR/xmlschema-1/#cElement_Declarations)

### 3.3.2 Deklaration von Attributen

Jede Schemadefinition kann durch die Deklaration von Attributen erweitert werden. Dies geschieht mithilfe des Elements `<xsd:attribute>`. Dieses Element kann innerhalb komplexer Datentypen verwendet werden, wobei es immer hinter den Elementdeklarationen erscheinen muss. In Listing 3.3.2.1 wird ein Attribut *isbn* eingefügt.

Listing 3.3.2.1: Beispiel für Attribute

```
<xsd:complexType name="computerType">
  <xsd:sequence>
    <xsd:element name="title" type="xsd:string"/>
    <xsd:element name="author" type="xsd:string"/>
  </xsd:sequence>
  <xsd:attribute name="isbn" type="xsd:token"/>
</xsd:complexType>
```

Die Datentypen des Attributs verwenden entweder die vorgegebenen einfachen Datentypen des XML-Schemas oder benutzerdefinierte einfache Datentypen. Attribute können also nicht geschachtelt werden oder andere Elemente beinhalten. Die Reihenfolge der Attributdeklarationen zu einem Datentyp ist beliebig. Die Namen der Attribute innerhalb eines komplexen Datentyps müssen aber eindeutig sein.

Auch für die Attribute können Default-Werte oder fixe Werte gesetzt werden, die unbedingt gelten, falls das Attribut verwendet wird:

```
<xsd:attribute name="format" type="xsd:token" default="PDF" use="optinal" />
```

Default-Werte werden bei Attributen verwendet, wenn das Attribut nicht angegeben ist, d.h., die Angabe von Default-Werten ergibt nur Sinn, wenn gleichzeitig mit *use="optinal"* gearbeitet wird.

```
<xsd:attribute name="format" type="xsd:token" fixe="PDF"/>
```

Wenn der Wert von *format* nicht verändert wird, wird ein fixer Wert als Attribut verwendet. Hier muss man beachten, dass Default-Werte und fixe Werte nicht gleichzeitig bei einem Attribut verwendet werden können.

Quelle: [http://www.w3.org/TR/xmlschema-1/#cAttribute\\_Declarations](http://www.w3.org/TR/xmlschema-1/#cAttribute_Declarations)

### 3.3.3 Globale und lokale Deklarationen

Globale Deklaration bedeutet, dass die Elemente `<xsd:element>` und `<xsd:attribut>` direkt unter das Element `<xsd:schema>` deklariert werden. Lokale Deklaration bedeutet,

dass die Elemente `<xsd:element>` und `<xsd:attribut>` innerhalb der komplexen Typdefinition deklariert werden. Bei einer lokalen Deklaration kann die Definition der globalen Deklaration durch das Attribut `ref` der Elemente `<xsd:element>` und `<xsd:attribut>` referenziert werden.

Listing 3.3.3.1: Beispiel für globale und lokale Deklarationen

```
<xsd:schema xmlns:xsd="http://www.w3.org/2001/XMLSchema">
  <xsd:element name="globalbook" type="bookType1"/>
  <xsd:element name="globaltitle" type="xsd:token"/>
  <xsd:attribute name="globalauthor" type="xsd:token"/>
  <xsd:complexType name="bookType1">
    <xsd:sequence>
      <xsd:element ref="globaltitle"/>
    </xsd:sequence>
    <xsd:attribute ref="globalauthor"/>
  </xsd:complexType>
  <xsd:complexType name="bookType2">
    <xsd:sequence>
      <xsd:element name="localtitle" type="xsd:token"/>
    </xsd:sequence>
    <xsd:attribute name="localauthor" type="xsd:token"/>
  </xsd:complexType>
</xsd:schema>
```

In Listing 3.3.3.1 sind die Deklarationen der Elemente *globalbook*, *globaltitle* und *globalauthor* globale Deklarationen, deshalb können die Elemente *globaltitle* und *globalauthor* von dem Attribut `ref` in der komplexen Typdefinition *bookType1* referenziert werden. Die Deklarationen der Elemente *localtitle* und *localauthor* sind lokale Deklarationen.

Folgende Ergänzungen gelten für die globale Deklaration:

- Der Name der globalen Deklaration von `<xsd:element>` und `<xsd:attribut>` muss im ganzen XSD-Dokument eindeutig sein.
- In der globalen Deklaration kann das Attribut `ref` nicht verwendet werden.
- Das Element der globalen Deklaration kann im Instanzdokument als Wurzelement verwendet werden.

Quelle: [http://www.w3.org/TR/xmlschema-1/#cElement\\_Declarations](http://www.w3.org/TR/xmlschema-1/#cElement_Declarations)  
[http://www.w3.org/TR/xmlschema-1/#cAttribute\\_Declarations](http://www.w3.org/TR/xmlschema-1/#cAttribute_Declarations)

### 3.3.4 Kompositoren

Die komplexen Elementtypen erlauben die Verwendung der Informationselemente als `<xsd:sequence>`, `<xsd:all>` und `<xsd:choice>`. Das Element `<xsd:sequence>` muss eine

vorgegebene Reihenfolge einhalten. Das Element `<xsd:all>` ermöglicht, dass die Elemente in der Gruppe in beliebiger Reihenfolge im enthaltenden Element angezeigt bzw. nicht angezeigt werden. Das Element `<xsd:choice>` lässt zu, dass nur genau eines der in der ausgewählten Gruppe enthaltenen Elemente im enthaltenden Element vorhanden ist.

**<xsd:sequence>** : Beim oberen Listing 3.3.1.2 ist es offensichtlich sinnvoll, die Einhaltung der Reihenfolge zu erzwingen. Das Element `title` muss vor dem Element `author` im Instanzdokument erscheinen.

**<xsd:all>** : In Listing 3.3.4.1 wird es nicht vorgeschrieben, welche Elemente vorhanden sein müssen, und auch die Reihenfolge wird offengelassen.

Listing 3.3.4.1: Beispiel für All

```
<xsd:complexType name="allType">
  <xsd:all>
    <xsd:element name="number" type="xsd:integer"/>
    <xsd:element name="name" type="xsd:token"/>
  </xsd:all>
</xsd:complexType>
```

Aber es muss beachtet werden, dass Modellgruppen mit `<xsd:all>` die Einschränkungen vorhanden sind. Sie müssen jeweils als einziges Kind einer Modellgruppe verwendet werden, können also nicht mit anderen Kindelementen `<xsd:sequence>` oder `<xsd:choice>` gemischt werden. Eine folgende Struktur ist nicht zulässig in Listing 3.3.4.2:

Listing 3.3.4.2: Beispiel für falsche All

```
<xsd:complexType name="falscheAllType">
  <xsd:sequence>
    <xsd:all>
      <xsd:element name="number" type="xsd:integer"/>
      <xsd:element name="name" type="xsd:token"/>
    </xsd:all>
    <xsd:sequence>
      <xsd:element name="age" type="xsd:positiveInteger"/>
      <xsd:element name="gender" type="xsd:token"/>
    </xsd:sequence>
  </xsd:sequence>
</xsd:complexType>
```

Die Kindelemente, die innerhalb von dem Element `<xsd:all>` sind, dürfen nur Einzelelemente und keine Gruppen verwendet werden. Auf der anderen Seite dürfen die Elemente innerhalb der Modellgruppe höchstens einmal erscheinen.

**<xsd:choice>** : Wenn einer von allen möglichen Werten ausgewählt wird, wird das

Element `<xsd:choice>` verwendet. In Listing 3.3.4.3 kann man entweder ein Telefon oder ein Handy auswählen.

Listing 3.3.4.3: Beispiel für Choice

```
<xsd:complexType name="choiceType">
  <xsd:choice>
    <xsd:element name="telefon">
      ...
    </xsd:element>
    <xsd:element name="handy">
      ...
    </xsd:element>
  </xsd:choice>
</xsd:complexType>
```

Die möglichen verschachtelten Gruppen kann man wie im folgenden Beispiel Listing 3.3.4.4 verwenden:

Listing 3.3.4.4: Beispiel für verschachtelte Gruppen

```
<xsd:complexType name="verschachtelteType">
  <xsd:sequence>
    <xsd:sequence>
      <xsd:element name="name" type="xsd:token"/>
      <xsd:element name="age" type="xsd:positiveInteger"/>
      <xsd:element name="gender" type="xsd:token"/>
    </xsd:sequence>
    <xsd:choice>
      <xsd:element name="telefon" type="xsd:token"/>
      <xsd:element name="handy" type="xsd:token"/>
    </xsd:choice>
  </xsd:sequence>
</xsd:complexType>
```

Quelle: [http://www.w3.org/TR/xmlschema-1/#Model\\_Groups](http://www.w3.org/TR/xmlschema-1/#Model_Groups)

### 3.3.5 Partikel

In den Kompositoren erscheint die Elementdeklaration nur einmal in dem Instanzdokument. Wenn die Elementdeklaration mehrere Male wiederverwendbar ist, kann man Partikel einsetzen. Ein Partikel kann die Zahlen des erscheinenden Elements kontrollieren. Hier gibt es zwei Attribute: das `minOccurs`-Attribut und `maxOccurs`-Attribut, die in der Modellgruppe [58] (`<xsd:sequence>`, `<xsd:all>` und `<xsd:choice>`) oder als Element `<xsd:element>` verwendet werden. Der Wert des `minOccurs`-Attributs ist der minimale Wert des erscheinenden Elements und der Wert des `maxOccurs`-Attributs ist der maximale Wert des erscheinenden Elements.

Default-Wert dieser zweiten Attribute ist 1. Beachtet werden muss aber, dass negative Werte nicht erlaubt sind, d.h., das Element kommt genau einmal vor, wenn keines der Attribute verwendet wird.

In Listing 3.3.5.1 kann man entweder zwei Telefone oder zwei Handys auswählen.

Listing 3.3.5.1: Beispiel für minOccurs-Attribut und maxOccurs-Attribut

```
<xsd:complexType name="minOccursmaxOccursType">
  <xsd:choice>
    <xsd:element name="telefon" type="xsd:token"
      minOccurs="1" maxOccurs="2"/>
    <xsd:element name="handy" type="xsd:token"
      minOccurs="1" maxOccurs="2"/>
  </xsd:choice>
</xsd:complexType>
```

Zusätzlich kann das maxOccurs-Attribut noch den Wert „unbounded“ annehmen, d.h., es gibt für diesen Wert keine Einschränkung.

Quelle: <http://www.w3.org/TR/xmlschema-1/#cParticles>

### 3.3.6 Mehrfache XML-Dokumente und - Namensräume

Wenn es darum geht, Datentypdefinitionen von außerhalb in ein Schema zu übernehmen und entweder unverändert oder in abgewandelter Form wiederzuverwenden, bieten die Elemente `<xsd:include>`, `<xsd:redefine>` und `<xsd:import>` ein mögliches Verfahren.

Das Element `<xsd:include>` kann mehrere Schemata inkludieren und der `targetNamespace` des `address.xsd` muss mit dem des inkludierenden Schemas übereinstimmen. Listing 3.3.6.1 zeigt ein Beispiel des Elements `<xsd:include>`.

Listing 3.3.6.1: Beispiel für include

```
<xsd:schema xmlns:xsd="http://www.w3.org/2001/XMLSchema"
  xmlns:emp="http://www.example.org/emp"
  targetNamespace="http://www.example.org/emp">
  <xsd:include schemaLocation="address.xsd"/>
  <xsd:complexType name="empType">
    <xsd:sequence>
      <xsd:element name="name" type="xsd:token"/>
      <xsd:element name="homeAddress" type="emp:addressType"/>
    </xsd:sequence>
    <xsd:attribute name="number" type="xsd:token"/>
  </xsd:complexType>
</xsd:schema>
```

Das Element `<xsd:redefine>` kann an Stelle von `<xsd:include>` verwendet werden und der Name des Typs ändert sich dabei nicht. Listing 3.3.6.2 zeigt ein Beispiel des Elements `<xsd:redefine>`.

Listing 3.3.6.2: Beispiel für redefine

```
<xsd:schema xmlns:xsd="http://www.w3.org/2001/XMLSchema"
  targetNamespace="http://www.example.org/emp"
  xmlns:emp="http://www.example.org/emp">
  <xsd:redefine schemaLocation="address.xsd">
    <xsd:complexType name="addressType">
      <xsd:complexContent>
        <xsd:extension base="emp:addressType">
          <xsd:sequence>
            <xsd:element name="zipcode" type="xsd:token"/>
          </xsd:sequence>
        </xsd:extension>
      </xsd:complexContent>
    </xsd:complexType>
  </xsd:redefine>
  <xsd:element name="employee" type="emp:empType"/>
  <xsd:complexType name="empType">
    <xsd:sequence>
      <xsd:element name="name" type="xsd:token"/>
      <xsd:element name="homeAddress" type="emp:addressType"/>
    </xsd:sequence>
    <xsd:attribute name="number" type="xsd:token"/>
  </xsd:complexType>
</xsd:schema>
```

Das Element `<xsd:import>` erlaubt es, Elemente aus anderen Namensräumen zu importieren, mit einem Präfix zu versehen und damit die Schema-Bestandteile aus unterschiedlichen Namensräumen wiederzuverwenden. Voraussetzung ist, dass es einen definierten Typ `addressType` in `address.xsd` gibt. Listing 3.3.6.3 zeigt ein Beispiel des Elements `<xsd:import>`.

Listing 3.3.6.3: Beispiel für import

```
<xsd:schema xmlns:xsd="http://www.w3.org/2001/XMLSchema"
  xmlns="http://www.sunxin.org/emp"
  targetNamespace="http://www.example.org/emp"
  xmlns:addr="http://www.example.org/address">
  <xsd:import schemaLocation="address.xsd"/>
  <xsd:element name="employee" type="empType"/>
```



```

<xsd:complexType name="empType">
  <xsd:sequence>
    <xsd:element name="name" type="xsd:token"/>
    <xsd:element name="homeAddress" type="addressType" />
  </xsd:sequence>
  <xsd:attribute name="number" type="xsd:token"/>
</xsd:complexType>
</xsd:schema>

```

Quelle: <http://www.w3.org/TR/xmlschema-1/#composition>

### 3.3.7 Identitätseinschränkungen

Hier werden die Elemente `<xsd:key>`, `<xsd:unique>` und `<xsd:keyref>` verwendet, um die Identität eines Elements in bestimmter Weise einschränken und für die Eindeutigkeit eines Elements sorgen zu können. Dabei wird eine eingeschränkte Menge von XPath-Ausdrücken eingebaut, die die Auswahl der „Schlüselfelder“ steuern [47]. Listing 3.3.7 bestätigt die Eindeutigkeit von „number“.

Listing 3.3.7: Beispiel für Identitätseinschränkung

```

<xsd:element name="company" type="companyType">
  <xsd:key name="numberUnique">
    <xsd:selector xpath="employee"/>
    <xsd:field xpath="number"/>
  </xsd:key>
</xsd:element>

```

Quelle: [http://www.w3.org/TR/xmlschema-1/#cidentity-constraint\\_Definitions](http://www.w3.org/TR/xmlschema-1/#cidentity-constraint_Definitions)

### 3.3.8 Definition von Element- und Attributgruppen

Die komplexe Elementtypen enthalten in der Regel Gruppierungen, in den mehrere Elemente und Attribute verwendet werden können. Diese Anwendungen sind durch eine Sequenz `<xsd:sequence>` oder eine Auswahlmöglichkeit `<xsd:choice>` realisiert.

**Elementgruppen:** Wenn die bestimmte Modellgruppen mehrfach benötigt werden, wird das Element `<xsd:group>` verwendet.

Listing 3.3.8.1 ist ein Beispiel, bei dem ein Teil der Informationen über Studenten zu einer Gruppe zusammengefasst wird. In dem anschließend definierten komplexen Typ `klasseType` wird die Gruppe mit dem Attribut `ref` eingefügt.

Listing 3.3.8.1: Beispiel für Elementgruppen

```
<xsd:group name="studentenGroup">
  <xsd:sequence>
    <xsd:element name="name" type="xsd:token"/>
    <xsd:element name="age" type="xsd:positiveInteger"/>
  </xsd:sequence>
</xsd:group>
<xsd:complexType name="klasseType">
  <xsd:group ref="studentenGroup"/>
</xsd:complexType>
```

**Attributgruppen:** Wenn mehrere Attribute zu einer Gruppe zusammengefasst werden, wird das Element `<xsd:attributeGroup>` verwendet. Diese Anwendung ist ähnlich wie Elementgruppen.

Listing 3.3.8.2 ist ein Beispiel, bei dem ein Teil der Informationen über eine Produktion zu einer Gruppe zusammengefasst wird. In dem anschließend definierten komplexen Typ `orderType` wird die Gruppe mit dem Attribut `ref` eingefügt.

Listing 3.3.8.2: Beispiel für Attributgruppen

```
<xsd:attributeGroup name="baseAttrGroup">
  <xsd:attribute name="number" type="xsd:token" use="required"/>
  <xsd:attribute name="totalPrice" type="xsd:decimal" use="required"/>
</xsd:attributeGroup>
<xsd:complexType name="orderType">
  <xsd:attribute Group ref="baseAttrGroup"/>
  <xsd:attribute name="orderTime" type="xsd:dateTime"/>
</xsd:complexType>
```

Quelle: [http://www.w3.org/TR/xmlschema-1/#cAttribute\\_Group\\_Definitions](http://www.w3.org/TR/xmlschema-1/#cAttribute_Group_Definitions)  
[http://www.w3.org/TR/xmlschema-1/#cModel\\_Group\\_Definitions](http://www.w3.org/TR/xmlschema-1/#cModel_Group_Definitions)

### 3.3.9 Ableitung komplexer Datentypen

Die Ableitungen komplexer Datentypen können wieder mit Erweiterungen oder Einschränkungen aufgrund einer vorhandenen Struktur arbeiten [66].

**Erweiterungen komplexer Elemente:** In Listing 3.3.1.1 ist ein komplexer Elementtyp `booklist` definiert. Es wird das Element `<xsd:complexContent>` verwendet. Listing 3.3.9.1 zeigt ein Beispiel, wenn der Wert des `star`-Elements „true“ ist, dann gehört das Buch zu dem Typ `starbooklist`. Das heißt, dass das im Element `<xsd:extension>` angegebene Element an die Sequenz der Elemente des Basistyps angehängt wird.

Listing 3.3.9.1: Beispiel für Vereinigungen

```

<xsd:complexType name="starbooklist">
  <xsd:complexContent>
    <xsd:extension base="booklist">
      <xsd:sequence>
        <xsd:element name="star" type="xsd:boolean"/>
      </xsd:sequence>
    </xsd:extension>
  </xsd:complexContent>
</xsd:complexType>

```

**Einschränkung komplexer Elemente:** In Listing 3.3.1.2 ist ein komplexer Elementtyp *computerType* definiert, hier wird das Element `<xsd:complexContent>` verwendet. Listing 3.3.9.2 zeigt ein Beispiel, bei dem das Element *author* durch `minOccurs="0"` nicht erscheint.

Listing 3.3.9.2: Beispiel für Vereinigungen

```

<xsd:complexType name="newcomputerType">
  <xsd:complexContent>
    <xsd:restriction base="computerType">
      <xsd:sequence>
        <xsd:element name="title" type="xsd:string"/>
        <xsd:element name="author" type="xsd:string" minOccurs="0" />
      </xsd:sequence>
    </xsd:restriction>
  </xsd:complexContent>
</xsd:complexType>

```

Quelle: [http://www.w3.org/TR/xmlschema-1/#Complex\\_Type\\_Definitions](http://www.w3.org/TR/xmlschema-1/#Complex_Type_Definitions)

### 3.3.10 Kommentare

XML-Schema bietet drei spezielle Elemente an, die für die Kommentierung eines Schemas genutzt werden können.

Das Element `<xsd:annotation>` ist ein Container, in den zusätzliche Informationen eingesetzt werden können, sei es für die Nutzung durch Menschen mit dem Element `<xsd:documentation>`, sei es für Programme mit dem Element `<xsd:appinfo>`. Das Element `<xsd:annotation>` kann fast jedem Element des XML-Schemas als erstes Kindelement hinzugefügt werden. Dieser Container kann auch überall als Element auf das Element `<xsd:schema>` angegeben werden. Aber es gibt eine Einschränkung. Das Element `<xsd:annotation>` kann nicht innerhalb von sich selbst auftreten [50].

Listing 3.3.10 zeigt ein Beispiel des Kommentars im XSD-Dokument.

Listing 3.3.10: Beispiel für Kommentare

```

<xsd:annotation>
  <xsd:appInfo>XSD4J Bibliothek </xsd:appInfo>
  <xsd:documentation xml:lang="en">
    Dieses Schema definiert die Kommentare der XSD4J Bibliothek!
  </xsd:documentation>
</xsd:annotation>
    
```

Quelle: <http://www.w3.org/TR/xmlschema-1/#cAnnotations>

### 3.4 Datentypen

Part 2: *Datetypes* in der XML-Schema-Spezifikation von W3C enthält ausführliche Beschreibung der Definieren der verwendeten Datentypen im XML-Schema. In diesem Teil werden integrierte primitive Datentypen, abgeleitete Datentypen und Facetten definiert.

Hier wird das Schema der eingebauten Datentypen wiedergegeben, das das W3C in den drei Teilen der XML-Schema-Spezifikation veröffentlicht hat.

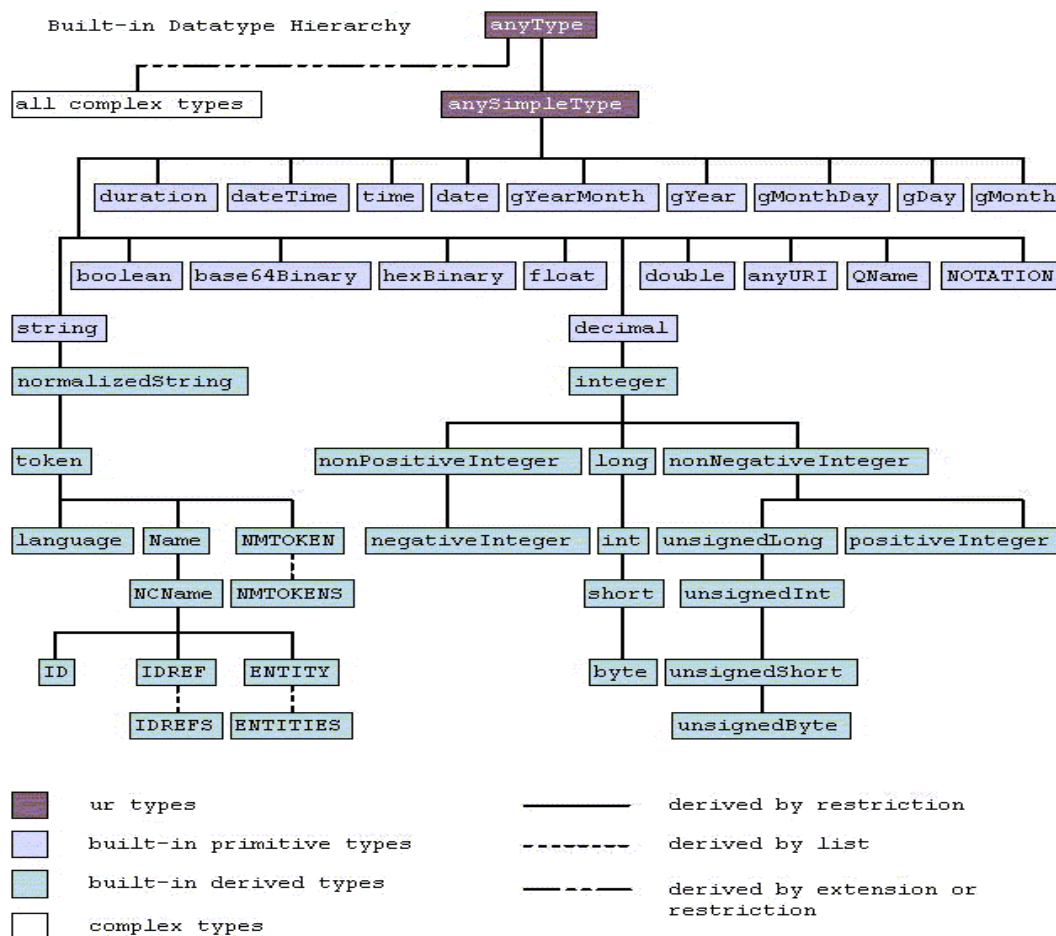


Abbildung 3.3: Hierarchie der Datentypen [8]

### 3.4.1 Komplexe Datentypen

Elemente, die selbst Unterelemente oder Attribute enthalten, werden in der Terminologie von XML-Schema komplexe Typen genannt [24]. Abschnitt 3.3.1 zeigt ein Beispiel.

### 3.4.2 Vordefinierte einfache Datentypen

Die Abbildung 3.3 zeigt die Hierarchie der Datentypen im XML-Schema. Die XML-Schema-Spezifikation empfiehlt 44 vordefinierte einfache Typen, die in Element- und Attribut-Deklarationen verwendet werden. Die vordefinierte einfache Datentypen bezeichnen, nur einen Wert, z.B. ein Datum, ein Text oder eine Zahl zu enthalten [51].

### 3.4.3 Primitive Datentypen

Die primitiven Datentypen haben 19 vordefinierte einfache Typen, sie können die folgenden sechs Formen annehmen:

#### I. Basis auf String Datentypen:

- anyURI:** Stellt einen Uniform Resource Identifier (URI) dar. Ein anyURI-Wert kann absolut oder relativ sein und darf Fragmentbezeichner enthalten, z.B. `http://www.tu-dresden.de`.
- string:** Zeichenketten in XML, z.B. „TUD“.

#### II. Numerische Datentypen:

- Float:** Stellt 32-Bit-Gleitkommazahlen mit einfacher Genauigkeit nach [IEEE 754-185] dar, z.B. `1.12e-2`.
- double:** Stellt 64-Bit-Gleitkommazahlen mit doppelter Genauigkeit nach [IEEE 754-185] dar, z.B. `123.123`.
- Decimal:** Stellt Dezimalzahlen beliebiger Genauigkeit dar, z.B. `-1.23`.

#### III. Datum und Zeit Datentypen:

- Date:** Stellt ein Kalenderdatum dar, z.B. `2010-04-29`.
- time:** Stellt eine Instanz der Zeit dar, die jeden Tag wiederkehrt.
- dateTime:** Stellt einen spezifischen Zeitpunkt dar, z.B. `2010-04-29T13:00:00-15:30:00`.
- gYear:** Stellt ein Jahr nach dem gregorianischen Kalender dar. Ein Satz nicht periodischer Instanzen, die ein Jahr dauern, z.B. `2010`.
- gYearMonth:** Stellt einen bestimmten Monat nach dem gregorianischen Kalender in einem bestimmten Jahr dar. Dabei handelt es sich um bestimmte, einen Monat lange, nicht periodische Instanzen, z.B. `2010-04`.
- gMonth:** Ein gregorianischer Monat, der jedes Jahr wiederkehrt, z.B. `04`.
- gMonthDay:** Tag nach dem gregorianischen Datum, genauer ein Tag eines Jahres, z.B. `04-29`.
- gDay:** Ein Tag in einem Monat nach dem gregorianischer Datum, z.B. `5`.
- duration:** Stellt eine Zeitdauer dar, z.B. `P1Y2M3DT12H10M`.

#### IV. Logischer Datentyp:

**Boolean:** Stellt boolesche Werte dar. Diese sind entweder true oder false (1 oder 0).

#### V. Binäre Datentypen:

**hexBinary:** hexadezimal kodierte Daten, z.B. 0FB7 für 4023.

**base64Binary:** Stellt beliebige Base64-codierte Binärdaten dar. base64Binary ist die Menge von Oktettsequenzen.

#### VI. Basis auf XML Datentypen:

**QName:** Stellt XML-qualifizierte Namen dar. QName ist eine Menge von Tupeln der Form( Lokale Teil im Namensraum), z.B. xsd:schema.

**NOTATION:** Name einer Notation.

### 3.4.4 Abgeleitete einfache Datentypen

Die primitiven Datentypen haben 25 vordefinierte einfache Typen, sie können den folgende sechs Formen zugeordnet werden. Im Folgenden sind die abgeleiteten XML-Schema Datentypen und eine kurze Beschreibung der abgeleiteten Datentypen erläutert [65].

#### I. Basis auf String Datentypen:

**Language:** Code für die natürlicher Sprachen (definiert durch RFC 1766), abgeleitet aus dem Typ *token*, z.B. de, cn.

**Name:** Namen in XML Ein Name ist ein Token, das mit einem Buchstaben, einem Unterstrich oder einem Doppelpunkt beginnt, auf den ein Namenszeichen (Buchstaben, Ziffern und andere Zeichen) folgt, abgeleitet aus dem Typ *token*, z.B. computer-book.

**NCName:** Ein Name, der keinen Doppelpunkt enthält. Dieser Datentyp entspricht dem Datentyp Name, mit dem Unterschied, dass er nicht mit einem Doppelpunkt beginnen darf, abgeleitet aus dem Typ *Name*, z.B. book.

**normalizedString:** Stellt durch Leerraum normalisierte Zeichenfolgen dar, abgeleitet aus dem Typ *string*.

**token:** Stellt aus Token bestehende Zeichenfolgen dar, abgeleitet aus dem Typ *normalizedString*, z.B. Auto.

**II. Die folgenden Datentypen werden bereitgestellt, um die Kompatibilität zwischen XML-Schema und DTDs zu wahren. Sie werden nur für die Attribute verwendet.**

**ENTITY:** Name einer allgemeinen Entität.

**ENTITIES:** Stellt den ENTITIES-Attributtyp dar. Enthält eine Gruppe von Werten vom Typ ENTITY.

**ID:** Eindeutiger Identifizierer eines Elements.

**IDREF:** Die Menge aller Zeichenketten, der BasisTyp sit NCName.

**IDREFS:** Die Menge von Token-Listen, deren Einträge durch Leerräume separiert sind.

**NMTOKEN:** Namen-Token.

**NMTOKENS:** Stellt den NMTOKENS-Attributtyp dar. Enthält eine Gruppe von Werten vom Typ NMTOKEN.

### III. Numerische Datentypen:

**Byte:** Ganzzahl von -128 bis 127, abgeleitet aus dem Typ *short*, z.B. 100.

**Int:** Ganzzahl von -2147483648 bis 2147483647, abgeleitet aus dem Typ *long*, z.B. 10.

**Integer:** Ganzzahl beliegiger Größe, abgeleitet aus dem Typ *decimal* durch Restriktion der Dezimalstellen auf 0, z.B. +1.

**Long:** Ganzzahl von -9223372036854775808 bis 9223372036854775807, abgeleitet aus dem Typ *integer*, z.B. 1069.

**negativeInteger:** Eine ganze Zahl, die kleiner als 0 ist, abgeleitet aus dem Typ *nonPositiveInteger*, z.B. -20.

**nonPositiveInteger:** Eine ganze Zahl, die kleiner oder gleich 0 ist, abgeleitet aus dem Typ *integer*, z.B. 0.

**nonNegativeInteger:** Eine ganze Zahl, die größer oder gleich 0 ist, abgeleitet aus dem Typ *integer*, z.B. 123.

**positiveInteger:** Eine ganze Zahl, die größer als 0 ist, abgeleitet aus dem Typ *nonNegativeInteger*, z.B. 120.

**Short:** Ganzzahl von -32768 bis 32767, abgeleitet aus dem Typ *int*, z.B. 1010.

**unsignedLong:** Ganzzahl von 0 bis 18446744073709551615 ohne Vorzeichen, abgeleitet aus dem Typ *nonNegativeInteger*, z.B. 2020.

**unsignedInt:** Ganzzahl von 0 bis 4294967295 ohne Vorzeichen, abgeleitet aus dem Typ *unsignedLong*, z.B. 725.

**unsignedShort:** Ganzzahl von 0 bis 65535 ohne Vorzeichen, abgeleitet aus dem Typ *unsignedInt*, z.B. 655.

**unsignedByte:** Ganzzahl von 0 bis 255 ohne Vorzeichen, abgeleitet aus dem Typ *unsignedShort*, z.B. 1.

### 3.4.5 Atomare Datentypen, Listen- und Vereinigungs-Datentypen

Die Ableitungen einfacher Datentypen sind abhängig von den bereits existierenden Datentypen als Grundlage. In der W3C-Empfehlung verteilen sich diese Datentypen auf drei Klassifikationsmöglichkeiten: Atomare Datentypen, Listen- und Vereinigungs-Datentypen.

**Atomare Datentypen** besitzen Werte, die bezüglich dieser Spezifikation als nicht weiter zergliederbar betrachtet werden [60], z. B. 2010, „TUD“, und 2010-04-30T23:59:59. Das heißt, atomare Datentypen enthalten eingebaute primitive Datentypen, abgeleitete eingebaute Datentypen und benutzerdefinierte einfache Datentypen. Eingebaute primitive Datentypen sind die von dem Urtyp `<xsd:anySimpleType>` direkt abgeleiteten Typen, wie der Typ *string* oder *decimal*. Die abgeleitete eingebauten Datentypen kommen aus beiden Teilen, wie der Typ *normalizedString* oder *integer*. Diese beide Teile sind im XML-Schema die Datentypen *string* und *decimal*. Benutzerdefinierte einfache Datentypen werden von den eventuell eingebauten primitiven Datentypen und abgeleiteten eingebauten Datentypen abgeleitet.

**Listen-Datentypen** erfolgt die Strukturierung der einzelnen Listeneinträge durch

Leerraumzeichen (whitespace) und nicht durch Elemente [69].

Listing 3.4.5.1 zeigt einen einfachen Datentyp *stadtType*, der durch Einschränkungen von `<xsd:string>` nur einen bestimmten Wert (Dresden, Shanghai) annehmen darf. Die Einschränkungen können eine Teilmenge von gültigen Werten aus der ursprünglichen Wertemenge festlegen.

Listing 3.4.5.1: Beispiel für Einschränkungen

```
<xsd:simpleType name="stadtType">
  <xsd:restriction base="xsd:string">
    <xsd:enumeration value="Dresden"/>
    <xsd:enumeration value="Shanghai"/>
  </xsd:restriction>
</xsd:simpleType>
```

Es lassen sich auch neue Datentypen durch Listen erzeugen, wie hier zum Listing 3.4.5.2 eine *stadtListe* von dem in Listing 3.4.5.1 erzeugten *stadtType*. Gültige Elemente zu diesem Datentyp würden immer mit Leerzeichen die einzelnen Listeneinträge voneinander trennen, beispielsweise "Dresden Shanghai" oder "Shanghai Dresden".

Listing 3.4.5.2: Beispiel für Listen

```
<xsd:simpleType name="stadtListe">
  <xs:list itemType="stadtType"/>
</xsd:simpleType>
```

**Vereinigungs-Datentypen** können die Vermischung von zwei Wertebereichen dargestellt. Listing 3.4.5.3 zeigt, dass der einfache Datentyp *vereinigung* für beliebig große nichtnegative Zahlen oder eine Zeichenkette „unbounded“ erlaubt wird.

Listing 3.4.5.3: Beispiel für den Vereinigungs-Datentyp

```
<xsd:simpleType name="vereinigung">
  <xsd:union>
    <xsd:simpleType>
      <xsd:restriction base='xsd:nonNegativeInteger'/>
    </xsd:simpleType>
    <xsd:simpleType>
      <xsd:restriction base='xsd:string'>
        <xsd:enumeration value='unbounded'/>
      </xsd:restriction>
    </xsd:simpleType>
  </xsd:union>
</xsd:simpleType>
```



### 3.4.6 Einschränkende Facetten für Datentypen

Einfache Typen (integrierte und abgeleitete Typen) enthalten Facetten. Eine Facette ist ein einzelner Definitionsaspekt, der die Festlegung einer Gruppe von Werten für einen einfachen Typ erleichtert. Length, minInclusive und maxInclusive sind beispielsweise gebräuchliche Facetten für die integrierten Datentypen. Alle Facetten für einfache Typen definieren den Satz gültiger Werte für diesen einfachen Typ.

Eine Facette wird als Element definiert. Jedes Facetten-Element weist ein fixed-Attribut auf, das einen booleschen Wert darstellt. Bei der Definition eines einfachen Typs kann verhindert werden, dass Ableitungen dieses Typs die Werte angegebener Facetten verändern. Dazu muss den Facetten das fixed-Attribut hinzu gefügt und dessen Wert auf „true“ festgelegt werden.

Die Facetten können einmal in einer Typdefinition verwendet werden, aber die Enumeration-Facette und die Pattern-Facette bilden die Ausnahmen. Sie können über mehrere Einträge verfügen und werden gruppiert.

Im folgenden Beispiel wird ein einfacher Typ veranschaulicht, für den das fixed-Attribut auf „true“ festgelegt wurde. Dadurch wird kein anderer Wert als 7 erstellt.

Listing 3.4.6: Die Facette

```
<xsd:simpleType name="Postcode">
  <xsd:restriction base="xsd:string">
    <xsd:length value="7" fixed="true"/>
  </xsd:restriction>
</xsd:simpleType>
```

### 3.4.7 Liste der einschränkenden Facetten

Die XML-Schema-Spezifikation 1.0 definiert 12 Facetten, die zur Einschränkung der Werte einfacher Typen verwendet werden können. Alle Facetten haben einen erforderlichen Wert und dieser Wert funktioniert für die gültige Arten aufgrund der verschiedenen Facetten. Die folgende Tabelle stellt die Bereiche der Facetten abhängig von den angewendeten Arten dar:

Art	Facette
Werteraum	minInclusive, maxInclusive, minExclusive, maxExclusive
Länge	length, minLength, maxLength
Genauigkeit	totalDigits, fractionDigits
Aufzählung	enumeration
Pattern	pattern
Leerraum	whiteSpace

Tabelle 3.1: Die verschiedenen Arten in den Facetten

Nachfolgend werden einschränkende Facetten für ihre Beschreibungen und die abgeleiteten Datentypen, für die sie gelten, aufgelistet.

Name	Beschreibung
enumeration	Angegebene Gruppe von Werten. Hierdurch wird ein Datentyp auf die angegebenen Werte beschränkt.
fractionDigits	Wert mit einer bestimmten Maximalanzahl von Dezimalstellen für den Nachkommateil.
length	Anzahl der Längeneinheiten. Längeneinheiten sind datentypabhängig. Dieser Wert muss ein nonNegativeInteger sein.
maxExclusive	Oberer Grenzwert (alle Werte sind kleiner als dieser Wert). Der Datentyp dieses Werts muss mit dem des geerbten Datentyps übereinstimmen.
maxInclusive	Maximaler Wert. Der Datentyp dieses Werts muss mit dem des geerbten Datentyps übereinstimmen.
maxLength	Maximale Anzahl der Längeneinheiten. Längeneinheiten sind datentypabhängig. Dieser Wert muss ein nonNegativeInteger sein.
minExclusive	Unterer Grenzwert (alle Werte sind größer als dieser Wert). Der Datentyp dieses Werts muss mit dem des geerbten Datentyps übereinstimmen.
minInclusive	Niedrigster Wert. Der Datentyp dieses Werts muss mit dem des geerbten Datentyps übereinstimmen.
minLength	Minimale Anzahl der Längeneinheiten. Längeneinheiten sind datentypabhängig. Dieser Wert muss ein nonNegativeInteger sein.
pattern	Spezielles Muster, dem die Werte des Datentyps entsprechen müssen. Hierdurch wird der Datentyp auf die Literale beschränkt, die dem angegebenen Muster entsprechen. Der pattern-Wert muss ein regulärer Ausdruck sein.
totalDigits	Wert mit einer bestimmten Maximalanzahl von Dezimalstellen.
whiteSpace	Der Wert muss entweder preserve, replace oder collapse lauten. Die whiteSpace-Facette kann für die meisten numerischen Datentypen nicht geändert werden.

Tabelle 3.2: Die Beschreibung der Facetten

Quelle : <http://www.w3.org/TR/2004/REC-xmlschema-0-20041028/#facetsTable1>

<http://www.w3.org/TR/2004/REC-xmlschema-0-20041028/#facetsTable2>

## 3.5 Vergleich mit anderen Schemasprachen

Alternativ zu XML-Schema gibt es verschiedene andere Ansätze für die Realisierung von Schematas. Die zwei populärsten Schemasprachen sind RELAX NG und Schematron.

### 3.5.1 RELAX NG

Regular Language Description for XML New Generation (RELAX NG) will vor allem eine sehr viel einfacher zu handhabende Alternative zu XML-Schema bieten. RELAX NG basiert auf RELAX und TREX. Ein RELAX-NG-Schema spezifiziert Muster für die Struktur und den Inhalt eines XML-Dokuments[25]. Dabei ist ein RELAX-NG-Schema selbst ein XML-Dokument, jedoch bietet es auch eine kompakte Nicht-XML-Syntax an.

RELAX NG hat gegenüber XML-Schema folgende Vorteile [26] :

- einfacher und dabei ausdrucksstärker (z.B. durch strukturelle Gleichbehandlung von Attributen und Elementen)
- leichter und intuitiver zu erlernen und zu lesen, auch für Nichtfachleute
- besser modularisierbar
- RELAX NG kann XML-Schema Part 2: Datatypes direkt benutzen und ist in dieser Hinsicht zu 100% kompatibel; zusätzlich existiert ein erweiterbares Typkonzept.
- RELAX NG ist mathematisch bzw. informationstheoretisch fundiert.

RELAX NG hat gegenüber XML-Schema jedoch folgende Nachteile [27] :

- keine Funktion der Vererbung
- Weil RELAX NG der Schemagültigkeitsprüfung gewidmet ist, kann es die Information der Applikation nicht für den Prozessor anbieten.

### 3.5.2 Schematron

Schematron ist eine Schemasprache zur Validierung von Inhalt und Struktur von XML-Dokumenten. Die Implementierung der Sprache wird über XSL-Transformationen realisiert. Schematron verwendet keine formale Grammatik, sondern findet Muster in der Dokumentstruktur. Dadurch ist es möglich, Regeln zu definieren, die mit herkömmlichen Schemasprachen, die auf Grammatiken basieren, nicht möglich wären. Dennoch sollte Schematron in erster Linie als Ergänzung, nicht als Konkurrenz zu anderen Schemasprachen verstanden werden [28].

Schematron hat gegenüber XML-Schema folgende Vorteile [29] :

- einfache Erlernbarkeit und Benutzung
- die Verwendung von XPath-Ausdrücken
- basiert auf der Schemagültigkeitsprüfung wie RELAX NG
- die Möglichkeit, eine Auswahl an Attributen anzugeben
- die Möglichkeit, Elemente und Attribute in Modellgruppen zusammenzufassen

- die Möglichkeit, das Inhaltsmodell abhängig vom Wert eines Elements oder Attributs zu ändern.

Schematron hat gegenüber XML-Schema jedoch folgende Nachteile [27] :

- Kann keine Instanzdatenmodelle anbieten. Deshalb kann man nicht nachvollziehen, welche Datenmodelle und Instanzdaten benutzt werden.
- Weil Schematron der Schemagültigkeitsprüfung gewidmet ist, kann es die Instanzinformation nicht transformieren, z.B. Datentypen und Default-Werte.
- Sofern er nicht ausdrücklich verboten ist, ist jeder Inhalt wirksam. Das erhöht das Fehlerrisiko des Autors.

### 3.6 Zusammenfassung

Im diesem Kapitel wurden zunächst die Inhaltsmodelle mit XML-Schema vorgestellt. Hier konnten nicht alle Feinheiten des XML-Schemas in ihrer Syntax und der Auswirkung auf die Dokumentmodellierung oder gar die 44 Datentypen mit ihren Facetten aus XML-Schema vorgestellt werden, die grundlegende Ausrichtung und Philosophie der Schemasprache wurde jedoch erläutert.

Dann wurden die zwei anderen Schemasprachen RELAX NG und Schematron kurz dargestellt, um XML-Schema mit ihnen vergleichen zu können. Der Vergleich zeigte, welche Sprache für welche Aufgaben mehr oder weniger gut geeignet ist.

## Kapitel 4

### Analyse der Architektur und Umsetzung von XSD4J

XSD4J<sup>1</sup> ist die Abkürzung der englischen „XML Schema library for Java“. Es wurde im Rahmen des Dynvocation Projekts [30], das ein Forschungsprojekt am Lehrstuhl für Rechnernetze der Fakultät Informatik an der Technischen Universität Dresden entsteht, erstellt. Weitere Informationen über dieses Projekt XSD4J bietet der Internetauftritt des Lehrstuhls.

Die XSD4J Bibliothek unterstützt derzeit in der SVN-Version (Standard: 1426) meist die existierten Merkmale wie einfache Typen, komplexen Typen, Einschränkung eines Typs, Elementdefinition und Attributedefinition. Das Ziel der XSD4J ist es, zur Analyse und Anwendung des XML-Schemas geschaffen zu werden. Einerseits ist es Ziel, eine hohe Konformität der XML-Schema-Spezifikation zu erreichen und unabhängig von den Auswirkungen auf andere Software zu sein, andererseits müssen WSGUI<sup>2</sup> Inferenzmechanismen besonders leicht implementierbar sein [31]. In dieser Arbeit wurde ein Servlet XSD4JTest unter dem Container Tomcat entwickelt. Dieses Test-Werkzeug wird im Kapitel 6 ausführlich erläutert.

#### 4.1 Die XSD4J Bibliothek

XSD4J ist eine XML-Schema Bibliothek, die verschiedenen Funktionen haben: die Überprüfung des XML-Schemas, die Validierung des Instanzdokuments, die Bildberechnung der Grafiken und die Umwandlung von regulären Ausdrücken nach XSD-Typen. XSD4J ist abhängig von *Regular Expressions Instances and XML Schema* (RegExInstantiator, *org.dynvocation.lib.regexinstant*) [57]. RegExInstantiator ist eine kleine Bibliothek mit regelmäßigem Umgang mit dem Ausdruck in dem spezifischen XML-Schema. Das entsprechende API-Dokument findet sich auf der Webseite<sup>3</sup>.

In dieser Arbeit ist die Hauptaufgabe die Umsetzung des Parsers des XML-Schemas in Java-Objekt und die anschließende Rückkehr zum Standard-XML-Schema-Baum.

##### 4.1.1 XML Parser

XML ist zwar zur Darstellung von Daten hervorragend geeignet, diese Daten müssen jedoch auch verarbeitet werden und dazu sind eines oder mehrere Programme erforderlich. Ein Vorteil beim Schreiben von XML-Anwendungen ist, dass es einen wiederverwendbaren Dienst implementiert. Beim XML-Parser (auch *Prozessor* genannt) wird die Textdarstellung eines Dokuments zerlegt und in eine Reihe von begrifflichen Objekten umgewandelt. Die Bibliothek XSD4J benutzt den DOMParser.

---

<sup>1</sup> Project XSD4J: <http://dynvocation.selfip.net/xsd4j>

<sup>2</sup> Standard Specification and Documentation: <http://wsgui.berlios.de/guidd/>

<sup>3</sup> Regular Expressions Instances and XML-Schema:  
<http://dynvocation.selfip.net/docs/api/regexinstant/>

Die XML-Spezifikation [32] schreibt zwingend vor, dass ein XML-Dokument wohlgeformt sein muss. In der Praxis bedeutet dies, dass ein XML-Parser beim kleinsten Fehler im Dokument aussteigt und eine Fehlermeldung liefert. Diese Genauigkeit von XML-Parsern hat einen wichtigen Grund [33]: Sie erleichtert die Implementierung von XML-Parsern, sodass diese sehr schlank sein können und wenig Speicherplatz brauchen.

### 4.1.2 DOM

Das Document Object Model (DOM)<sup>4</sup> ist eine wesentliche plattform- und sprachunabhängige Programmierschnittstelle (API), die vom W3C definiert und eine genormte Schnittstelle ist, um XML-Dokumente zu behandeln. DOM ermöglicht es externen Programmen, dynamisch auf den Inhalt, die Struktur und das Layout von XML-Dokumenten zuzugreifen und sie zu verändern [78].

DOM-Normen nehmen Bezug auf ein Dokument, in dem ein Objektmodell nach dem Modell "basiert auf dem Baum" definiert wird. Dieses Objektmodell kann ein strukturiertes Dokument in der Form von Bäumen ausdrücken, sodass jedes Datenelement eines Dokumentes innerhalb des DOM eindeutig adressierbar ist. Die Daten können ggf. weiterverarbeitet und die Verarbeitungsergebnisse wieder in die aktuelle Seite eingebunden werden

Das XML-Parser basiert auf DOM bekannt ist. DOM basiert auf Bäumen (tree based) XML-Parser, weil seine hauptsächliche Idee aus dem Gesetz, das ein vollständiger Parser einer einmaligen Struktur des Dokuments ist. Anschließend wird ein Objekt-Baum verwendet, um das Dokument zu beschreiben. Am Ende sind alle Operationen (z.B. Modifizieren, Suchen, usw.) in dem Objekt-Baum manipuliert [75]. Die XSD4J Bibliothek ist abhängig von DOM, das XML-Dokument zu behandeln.

### 4.1.3 XSD4J mit dem DOM

Die Implementierung der XSD4J Bibliothek ist abhängig von DOM. Der Parser in der XSD4J Bibliothek verwendet auch DOMParser von dem XSD-Dokument in Java Objekte sowie Dumper von Java Objekte in XSD-Dokument.

Hier liegt allerdings auch der wesentliche Nachteil von DOM: Für sehr große XSD-Dokumente ist es unter Umständen nicht nutzbar. Wenn ein Programm beispielsweise sehr schnell nur das erste Subelement der Wurzel benötigt, muss mit dem DOM-Ansatz dennoch das gesamte Dokument eingelesen und geparkt werden [34].

DOM repräsentiert, wie bereits erwähnt, ein XML-Dokument als einen Baum von Knoten, wobei jeder Knoten die Wurzel eines darunter anschließenden Unterbaums sein kann.

Das hat u.a. den Vorteil im Parser, dass man alle Knoten einheitlich behandeln kann, er die Wurzel startet und sich selbst jeden Unterknoten der Wurzel aufrufen kann.

---

<sup>4</sup>W3C DOM Interest Group: <http://www.w3.org/DOM/>

## 4.2 Architektur von XSD4J

Die Abbildung 4.1 zeigt die Architektur der XSD4J Bibliothek. XSDSchema ist eine Zentrale der XSD4J Bibliothek. Das Ziel der XSDParser realisiert die Umsetzung von XSD-Dokument nach XSDSchema. Die Funktion von XSDDumper ist genau das Gegenteil von XSDParser. Das Ziel der XSDDumper realisiert die Umsetzung von XSDSchema in ein XSD-Dokument. Außerdem sind XSDTransformer, XSDInstantiator und XSDValidator auch abhängig von XSDSchema. W3C stellt geeignete Interfaces bereit. Für die hier benötigte Anwendung gibt es einen sogenannten DOMParser, um das originale Dokument in ein sogenanntes Dokument Objekt gemäß der Schnittstelle zu transformieren. XML-Dokument und XSD-Dokument können von DOMParser und XSDInstantiator instanziiert werden.

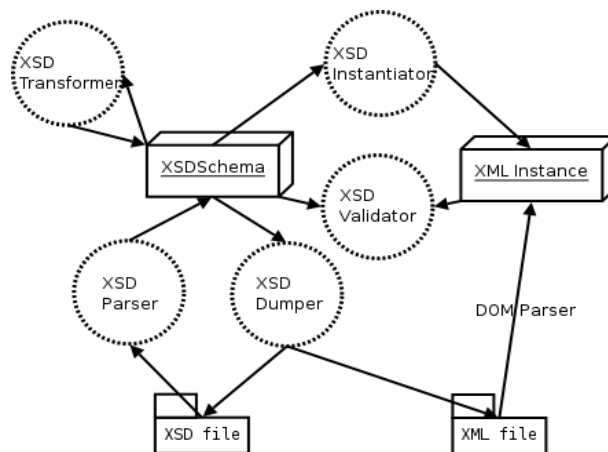


Abbildung 4.1: Die Architektur der XSD4J Bibliothek [35]

Tabelle 4.1 beschreibt die Bibliotheken, Klassen und Dokumente in der XSD4J Bibliothek

Name	Teil	Beschreibung
Bibliotheken	XSDSchema	Definition von XSD Objekte und Verwendung in den verschiedenen Levels flat, flatincludes, tree, und merged.
	XML Instance	Instanzen des XML-Dokuments
Klassen	XSDParser	Parser des XSD-Dokuments nach dem Java-Objekt
	XSDDumper	Dumper von Java-Objekt nach dem Standard XSD Dokument.
	XSDTransformer	Transformer zwischen Java Objekte(XSDSchema)
	XSDInstantiator	Instantiator von XSDSchema nach XML Instanz.
	XSDValidator	Validierung zwischen XMLSchema-Dokument und XML Instanzendokument.

	DOMParser	Ein XML-Parser liest das XML-Dokument und wandelt es zu einem XML-DOM-Objekt um.
Dokumente	XSD Dokument	Definieren von den Strukturen für das XML-Dokument.
	XML Dokument	Das XML-Dokument ist ein Instanzdokument gegenüber das XSD-Dokument.

Tabelle 4.1: Die Bibliotheken, Klassen und Dokumente in der XSD4J Bibliothek

### 4.2.1 APIs in der XSD4J Bibliothek

Die APIs sind nach dem Package geliedert. Innerhalb der Packages sind Klassen oder Interfaces ohne Berücksichtigung der Groß- und Kleinschreibung sortiert. Folgende Packages sind aufgeführt:

Package	Beschreibung
<i>org.dynvocation.lib.regexinstant</i>	Dieses Package enthält die Klassen und Interfaces für regelmäßigen Umgang mit den Ausdrücken in den spezifischen XML-Schema <i>regexp</i> Syntax. Es wird intern von XSD4J verwendet.
org.dynvocation.lib.xsd4j	Dieses Package enthält verschiedene Klassen, die die Funktionen der XSD4J Bibliothek implementieren. Die hauptsächlichen Funktionen sind: XSDParser, XSDDumper, XSDTransformer, XSDInstantiator und XSDValidator.

Tabelle 4.2: Die Packages in der XSD4J Bibliothek



Die einzelnen Klassen von XSDParser werden folgendermaßen dargestellt:

Methode von XSDParser	Beschreibung
<i>addNamespace</i>	<i>public void addNamespace(String prefix, String namespace)</i> Diese Methode liest <i>prefix</i> und <i>namespace</i> aus den Namensräumen des geparsten XSD-Dokuments auf <i>HashMap namespace</i> .
<i>addSchemaElement</i>	<i>public void addSchemaElement(Element schema, String originlocation)</i> Diese Methode wird vor dem Parser verwendet, wenn andere XML-Fragmente zusätzliche Informationen für das Schema enthalten.
<i>addSchemaFile</i>	<i>public void addSchemaFile(String schemafilename)</i> Diese Methode addiert die XSD-Dokumente zu parsen.
<i>stringToBigInteger</i>	<i>private BigInteger stringToBigInteger(String s)</i> Diese Methode kann den Datentyp von <i>String</i> in <i>BigInteger</i> umsetzen.
<i>elementlist</i>	<i>private ArrayList elementlist(Element parent, String filter)</i> Die Auflistung der geparsten Elemente im XSD-Dokument.
<i>evaluateXPath</i>	<i>private NodeList evaluateXPath(String xpathString, Element el)</i> Diese Methode berechnet den Wert des XPath-Ausdrucks und gibt das Ergebnis zum Element zurück.
<i>getNamespaces</i>	<i>public HashMap getNamespaces()</i> Diese Methode erhält die Namensräume des geparsten XSD-Dokuments.
<i>loadIncludesImports</i>	<i>private XSDSchema loadIncludesImports(XSDSchema xsdschema, int level, ArrayList xsds)</i> Diese Methode parst die Elemente <i>&lt;xsd:include&gt;</i> und <i>&lt;xsd:import&gt;</i> des XSD-Dokuments.
<i>localPart</i>	<i>private String localPart(String s)</i> Diese Methode erhält die lokale Präfixe des Namensraums
<i>multiparser</i>	<i>private XSDSchema multiparser(int level)</i> Diese Methode parst das XSD-Dokument nach dem ausgewählten Level.
<i>prefixPart</i>	<i>private String prefixPart(String s)</i> Diese Methode erhält die Präfixe des Namensraums oder <i>targetnamespace</i> .
<i>preload</i>	<i>private void preload()</i> Diese Methode exportiert die Informationen des geparsten XSD-Dokuments, z.B. Namensräume, Namen und die Elemente des XSD-Dokuments.

<i>qname</i>	<i>private QName qname(String typestr, boolean use_tns)</i> Diese Methode entscheidet, ob ein Präfix von <i>targetnamespace</i> in den Namensräumen des XSD-Dokuments existiert.
<i>parseXSDAlternative</i> <sup>5</sup>	<i>private XSDAlternative parseXSDAlternative(Element el)</i> Diese Methode weist dem Parser das Attribut <code>&lt;xsd:alternativ&gt;</code> im XSD-Dokument zu.
<i>resetLocal</i>	<i>private void resetLocal()</i> Definition der standardisierten Werte des XSD-Dokuments
<i>rootnamespace</i>	<i>private String rootnamespace(String prefix)</i> Diese Methode erhält die Namensraum anhand der Präfix in den Namensräumen, die aus dem geparsten XSD-Dokument auf HashMap <i>namespace</i> entstammen.
<i>stats</i>	<i>private void stats(XSDSchema xsdschema)</i> Diese Methode legt eine Statistik über das XSD-Dokument, z.B. wie viele XSDs, welches Level und die Strukturen des XSD-Dokuments. Sie ruft die Methode <i>stats_internal</i> auf.
<i>stats_internal</i>	<i>private void stats_internal(XSDSchema xsdschema, int layer)</i> Eine Rekursion für die Exporte des Schemas
<i>XPathEvaluator</i>	<i>private void XPathEvaluator(String xmlFilename)</i> Die Umsetzung von XML in DOM Baum.

Tabelle 4.3: Die Methoden in XSDParser

---

<sup>5</sup>Die Methode, die den Beginn der Zeichenkette „parseXSD“ enthält, parst die entsprechenden Elemente und Attribute des XSD-Dokuments, z.B. die Methode *parseXSDAlternative*, die den Parser des Attributs *Aternative* im XSD-Dokument realisiert.

Die einzelnen Klassen von XSDDumper werden folgendermaßen dargestellt:

Methode Von XSDDumper	Beschreibung
<i>basicSchema</i>	<i>private Element basicSchema(Document doc, XSDSchema xsdschema, String namespace)</i> Dumpen die Attribute des Elements <code>&lt;xsd:schema&gt;</code> .
<i>bigIntegerToString</i>	<i>private String bigIntegerToString(BigInteger x)</i> Diese Methode kann den Datentyp <i>BigInteger</i> in den Datentyp <i>String</i> umsetzen
<i>convert</i>	<i>public Element convert(XSDSchema xsdschema)</i> Diese Methode realisiert die Umsetzung vom Java Objekt des XSDSchemas in ein XML-Element. Sie ist abhängig von den Methoden <i>basicSchema</i> und <i>definitions</i> .
<i>convertElement</i>	<i>public Element convertElement(XSDElement xsdelement, XSDSchema xsdschema)</i> Diese Methode realisiert die Umsetzung von Java Objekt des XSDElements in ein XML-Element. Der Namensraum wird <i>targetnamespace</i> . Sie ist abhängig von der Klasse <i>XSDTransformer</i> und der Methode <i>definitions</i> .
<i>definitions</i>	<i>private void definitions(Document doc, Element parent, XSDSchema xsdschema)</i> Dumpen alle Inhalte des Schemas
<i>dump</i>	<i>public String dump(XSDSchema schema)</i> Diese Methode realisiert die Umsetzung von Java Objekt des XSDSchemas in den Text, dann kehrt der Text als Datentyp <i>String</i> zurück. Sie ist abhängig von den Methoden <i>convert</i> und <i>dumpXML</i> .
<i>dumpdom</i>	<i>private void dumpdom(Element el, boolean outermost)</i> Diese Methode setzt den Datentyp <i>String</i> des Elements in das XML-Format um. Sie ist abhängig von der Methode <i>output</i> .
<i>dumpElement</i>	<i>public String dumpElement(XSDElement xsdelement, XSDSchema xsdschema)</i> Diese Methode ist ähnlich wie <i>dump</i> und sie setzt nur ein einziges Element um, das für die Nachrichten basis auf dem Element geeignet ist. Die Erklärung dieses Elements braucht in der Ausgabe nur die Teile des Schemas zu enthalten. Sie ist abhängig von den Methoden <i>convert</i> und <i>dumpXML</i> .
<i>dumpXML</i>	<i>public String dumpXML(Element el)</i> Diese Methode erstellt ein standardisiertes XSD-Dokument. Sie ist abhängig von der Methode <i>dumpdom</i> .
<i>output</i>	<i>private void output(String s)</i>

	Diese Methode akzeptiert das Ergebnis der Methode <i>dumpdom</i> .
<i>scannamespaces</i>	<i>private void scannamespaces(Element el)</i> Diese Methode prüft das Element, ob der Datentyp des Elements die Präfixe der Namensräume enthält oder nicht.
<i>tagname</i>	<i>private String tagname(Node n)</i> Diese Methode stellt einen Knotenname für <i>tag</i> wieder her.
<i>xsdump_alternative</i> <sup>6</sup>	<i>xsdump_alternative(XSDAlternative xsdalter, Document doc, Element parent)</i> Diese Methode erstellt den Dumper des Elements <code>&lt;xsd:alternative&gt;</code> des XSD-Dokuments

Tabelle 4.4: Die Methoden in XSDDumper

## 4.2.2 Der Implementierungsablauf der Klasse XSDParser

Die Klasse XSDParser kommt aus dem Package *org.dynvocation.lib.xsd4j*. Sie analysiert XSD-Dokumente nach den verschiedenen Levels *flat*, *flatincludes*, *tree* und *merged*. Default-Level ist *flat*. Listing 4.2.2.1 zeigt ein Beispiel für das XSD-Dokument des Elements `<xsd:schema>`, das ein Unterelement `<xsd:annotation>` hat.

Listing 4.2.2.1: annotation.xsd

```

<?xml version="1.0" encoding="UTF-8"?>
<xsd:schema xmlns:xsd="http://www.w3.org/2001/XMLSchema"
  <xsd:annotation>
    <xsd:documentation>
      documentation content 1
    </xsd:documentation>
    <xsd:appinfo> appinfo. content 1 </xsd:appinfo>
    <xsd:documentation source="http://linked-documentation.invalid">
      documentation content 2
    </xsd:documentation>
    <xsd:appinfo source="http://linked-appinfo.invalid">
      appinfo content 2
    </xsd:appinfo>
  </xsd:annotation>
  ...
</xsd:schema>

```

<sup>6</sup>Die Methode, die den Beginn der Zeichenkette „xsdump“ enthält, dumpt die entsprechenden Elemente und Attribute des XSD-Dokuments, z.B. die Methode *xsdump\_alternative*, die den Dumper des Attributs *Alternative* im XSD-Dokument realisiert.

Der Parser wendet das Package *org.w3c.dom* an [73]. Dieses Beispiel beschreibt den Parser des Elements `<xsd:annotation>` im XSD-Dokument. Die Abbildung 4.2 illustriert den Ablauf des Parsers des Elements `<xsd:annotation>` in der Klasse `XSDParser` (von 1 bis zu 4).

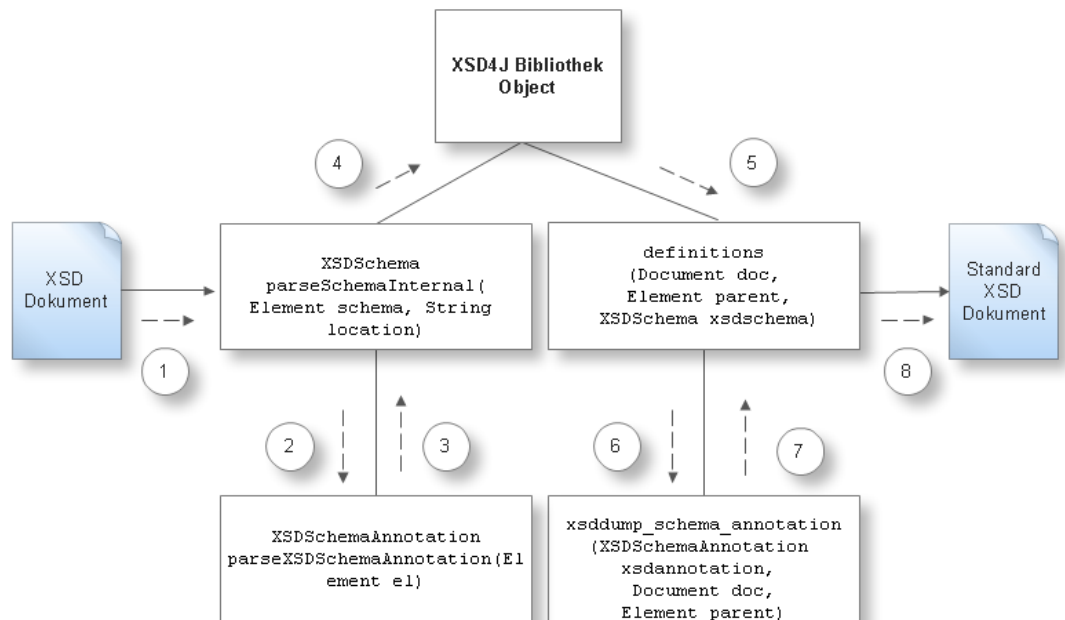


Abbildung 4.2: XSDParser für das Element Annotation

Die Struktur des Elements `<xsd:annotation>` wird von der W3C-Empfehlung<sup>7</sup> in Listing 4.2.2.2 vorgestellt:

Listing 4.2.2.2: Die Struktur des Elements Annotation

```
<annotation
  id = ID
  {any attributes with non-schema namespace . . .}>
  Content: (appinfo | documentation)*
</annotation>
```

Aufgrund der Struktur des Element `<xsd:schema>`<sup>8</sup> werden die Eigenschaften im Element `<xsd:schema>` definiert. Die Implementierung zeigt Listing 4.2.2.3. Wenn der Parser das Element `<xsd:schema>` analysiert, dann findet der Parser auch das Unterelement `<xsd:annotation>`. Danach ruft der Parser die Methode `parseXSDAnnotation` auf, um das Unterelement `<xsd:annotation>` umzusetzen.

<sup>7</sup><http://www.w3.org/TR/2009/WD-xmlschema11-1-20091203/#element-annotation>

<sup>8</sup><http://www.w3.org/TR/2009/WD-xmlschema11-1-20091203/#element-schema>

Listing 4.2.2.3: Aufruf der Methode *parseXSDAnnotation* in der Klasse XSDParser

```

...
NodeList list = el.getChildNodes();
    for (int i = 0; i < list.getLength(); i++) {
        Node node = list.item(i);
        if (node.getNodeType() != Node.ELEMENT_NODE) {
            continue;
        }
        String name = node.getLocalName();
        if (name.equals("annotation")) {
            debug("+ Found alternative annotation");
            XSDAnnotation xsdann = parseXSDAnnotation((Element) node);
            xsdalter.setAnnotation(xsdann);
        } else {
            debug("Warning: unknown alternative tag " + name);
            if (this.strict)
                throw new ParseException(null, 0);
        }
    }
...

```

Die Methode *parseXSDAnnotation* definiert die Beschreibung der Struktur *<xsd:annotation>* mit dem Parameter (Element *el*) in Listing 4.2.2.4. Die Variable *el* ist normalerweise das Elternelement. Im Beispiel ist es das Element *<xsd:schema>*. Die Methode *parseXSDAnnotation* enthält die Implementierung der Unterelemente *<xsd:document>* und *<xsd:appinfo>* sowie deren Strukturen.

Listing 4.2.2.4: Die Methode *parseXSDAnnotation* in der Klasse XSDParser

```

private XSDAnnotation parseXSDAnnotation (Element el) throws ParseException {
    XSDAnnotation xsdann = new XSDAnnotation();
    NodeList list = el.getChildNodes();
    for (int i = 0; i < list.getLength(); i++) {
        Node node = list.item(i);
        if (node.getNodeType() != Node.ELEMENT_NODE) {
            continue;
        }
        String name = node.getLocalName();
        if (name.equals("documentation")) {
            debug("find element documentation...");
            Element documentation = (Element) node;
            String urlstr = documentation.getAttribute("source");

```

```

        String docstr = textvalue(node);
        if (urlstr.length() > 0) {
            xsdann.setDocumentationURL(urlstr);
        }
        if (docstr.length() > 0) {
            xsdann.setDocumentation(docstr);
        }
        debug("element documentation end...");
    } else if (name.equals("appinfo")) {
        debug("find element appinfo...");
        Element appinfo = (Element) node;
        String urlstr = appinfo.getAttribute("source");
        String docstr = xmlcontents(node);
        if (urlstr.length() > 0) {
            xsdann.setAppInfoURL(urlstr);
        }
        if (docstr.length() > 0) {
            xsdann.setAppInfo(docstr);
        }
        debug("element appinfo end...");
    } else {
        debug("Warning: unknown annotation tag " + name);
        if (this.strict)
            throw new ParseException(null, 0);
    }
}
return xsdann;
}

```

Zusammenfassung des Implementierungsablaufs von XSDParser (von 1 bis zu 4):

1. Die Methode *parseSchemaInternal* liest ein XSD-Dokument oder mehrere XSD-Dokumente, um sie auf den Parser vorzubereiten. Das ist abhängig von der Methode *parseXSDSchemaAnnotation*.
2. Die Methode *parseSchemaInternal* ruft die Methode *parseXSDSchemaAnnotation* auf, um den Parser des Elements `<xsd:annotation>` im XSD-Dokument zu analysieren.
3. Nachdem die Methode *parseXSDSchemaAnnotation* den Parser des Elements `<xsd:annotation>` im XSD-Dokument analysiert hat, kehrt sie das Ergebnis des Parsers nach der Methode *parseSchemaInternal* zurück.
4. Am Ende sendet die Methode *parseSchemaInternal* das Ergebnis des Parsers im komplexen XSD-Dokument zur XSD4J Bibliothek.

### 4.2.3 Der Implementierungsablauf der Klasse XSDDumper

Die Klasse XSDDumper kommt aus dem Package *org.dynvocation.lib.xsd4j*. Sie analysiert Java-Objekte in der XSD4J Bibliothek nach den verschiedenen XSD-Spezifikationen XSD 1.0 (BootstrappedSchema10) und XSD 1.1 (BootstrappedSchema11). Listing 4.2.3.1 zeigt ein Beispiel für das XSD-Dokument des Elements `<xsd:schema>`. Wenn es ein Unterelement `<xsd:annotation>` im XSD-Dokument gibt, ruft XSDDumper die Methode `xsd_dump_annotation` auf.

Listing 4.2.3.1: Aufruf der Methode `xsd_dump_annotation` in der Klasse XSDDumper

```
XSDAnnotation xsdanno = xsdalter.getAnnotation();
    if(xsdanno !=null){
        xsddump_annotation(xsdanno, doc, xsd_alt);
    }
```

Durch den Aufruf der Methode im entsprechenden Element kann das Element, das mit dem Elements `<xsd:annotation>` beschrieben werden kann, implementiert werden. Die folgende Programmierung wird aufgerufen:

Die Methode `xsd_dumper_annotation` definiert die Beschreibung der Struktur `<xsd:annotation>` in Listing 4.2.3.2. Die Variable `xsdannotation` ist ein Java-Objekt (XSDAnnotation) und sie bekommt die abgebildeten Dateien nach dem Parser des XSD-Dokuments in der XSD4J Bibliothek. Die Methode `xsd_dumper_annotation` enthält die Implementierung der Unterelemente `<xsd:document>` und `<xsd:appinfo>` sowie deren Strukturen.

Listing 4.2.3.2: Die Methode `xsd_dumper_annotation` in der Klasse XSDDumper

```
private void xsddump_annotation (XSDAnnotation xsdannotation, Document
doc,Element parent) {
    if (xsdannotation == null){
        return;
    }
    String ns_xsd = XSDCommon.NAMESPACE_XSD;
    Element xsd_annotation = doc.createElementNS(ns_xsd, "annotation");
    parent.appendChild(xsd_annotation);
    if ((xsdannotation.getDocumentation() != null)
        || (xsdannotation.getDocumentationURL() != null)) {
        ArrayList textlist = xsdannotation.getDocumentation();
        for (int i = 0; i < textlist.size(); i++) {
            String text = (String)textlist.get(i);
            Element xsd_documentation = doc.createElementNS(ns_xsd,
                "documentation");
            xsd_annotation.appendChild(xsd_documentation);
```



```

        if (xsdannotation.getDocumentationURL() != null) {
            xsd_documentation.setAttribute("source",xsdannotation
                .getDocumentationURL().get(i).toString());
        }
        if (xsdannotation.getDocumentation() != null) {
            Text content = doc.createTextNode(text);
            xsd_documentation.appendChild(content);
        }
    }
}
if ((xsdannotation.getAppInfo() != null)
    || (xsdannotation.getAppInfoURL() != null)) {
    ArrayList textlist =xsdannotation.getAppInfo();
    for (int i = 0; i < textlist.size(); i++) {
        String text = (String)textlist.get(i);
        Element xsd_appinfo = doc.createElementNS(ns_xsd, "appinfo");
        xsd_annotation.appendChild(xsd_appinfo);
        if (xsdannotation.getAppInfoURL() != null) {
            xsd_appinfo.setAttribute("source", xsdannotation
                .getAppInfoURL().get(i).toString());
        }
        if (xsdannotation.getAppInfo() != null) {
            Text content = doc.createTextNode(text);
            xsd_appinfo.appendChild(content);
        }
    }
}
}
}

```

Die Zusammenfassung des Ablaufs von XSDDumper (von 5 bis 8) :

5. Die Methode *definitions* liest die Java-Objekte, um sie auf den Dumper vorzubereiten. Das ist abhängig von der Methode *xsdump\_schema\_annotation*.
6. Die Methode *definitions* ruft die Methode *xsdump\_schema\_annotation* auf, um den Dumper des Elements `<xsd:annotation>` im Java-Objekt zu analysieren.
7. Nachdem die Methode *xsdump\_schema\_annotation* den Dumper des Elements `<xsd:annotation>` im Java-Objekt analysiert hat, kehrt sie zum Ergebnis des Dumpers nach der Methode *definitions* zurück.
8. Am Ende sendet die Methode *definitions* das Ergebnis des Dumpers, welches ein standardisiertes XSD-Dokument ist.

### 4.3 Zusammenfassung

Im diesem Kapitel wurde die XSD4J Bibliothek detaillierter vorgestellt. Es handelt sich um das Projekt Dynvokation, RegExInstantiator und um die Struktur der XSD4J Bibliothek sowie die APIs von Parser und Dumper.

Hier wurden hauptsächlich die Klassen von XSDParser und XSDDumper beschrieben. Außerdem lieferte das Kapitel ein Beispiel mit dem Element `<xsd:annotation>` im XSD-Dokument. Dieses Beispiel stellte die Abläufe von Parser und Dumper des XSD-Dokuments in der Implementierung der XSD4J Bibliothek dar.

Zur Vervollständigung der XSD4J Bibliothek wurde Folgendes implementiert:

XSD4J hat bisher die Elemente und Attribute meist anhand der XSD 1.1 Spezifikation unterstützt. Diese Elemente und Attribute enthalten *Assertions*, *conditional type assignment* (CTA), die fünf neuen Datentypen, *open content*, und das Attribut *inheritable* von `<xsd:attribute>`, *defaultAttributesApply* des komplexen Typs, die neuen Regeln für die Einschränkung des komplexen Typs und die vier neuen Facetten sowie Grossteile des Elements `<xsd:override>`. Außerdem kann das Element `<xsd:any>` im Element `<xsd:all>` verwendet werden und die Teile von XSD 1.0 wie das Element `<xsd:annotation>` wurden vervollständigt.

Diese fünf neuen Datentypen sind `<xsd:anyAtomicType>`, `<xsd:precisionDecimal>`, `<xsd:dateTimeStamp>`, `<xsd:yearMonthDuration>` und `<xsd:dayTimeDuration>`, sie die Datenstrukturen der XSD4J Bibliothek eingeführt werden. Die vier neuen Facetten, die `<xsd:assertion>`, `<xsd:explicitTimezone>`, `<xsd:minScale>` und `<xsd:maxScale>` sind, werden in die Parser- und Dumperklasse addiert. Die Änderungen anderer Strukturen werden in der Tabelle B2 im Anhang dargestellt.

Die Implementierung automatisch überprüfbarer Testfälle und überprüfbares Einzelfalls beschreibt Kapitel 6.

# Kapitel 5

## Arbeit mit XSD 1.1

Dieses Kapitel wird zeigen, welche neuen Strukturen und Datentypen in XSD 1.1 vorhanden sind. Die einzelnen Strukturen und Datentypen werden hier jeweils durch Beispiele illustriert. Die letzte Entwicklung der XML Schema Working Group (XSWG) für die Arbeitsentwürfe W3C XML Schema Definition Language (XSD) 1.1<sup>1</sup> besteht aus zwei Teilen: *Part 1: Structures*<sup>2</sup> und *Part 2: Datatype*<sup>3</sup>. Diese bisherigen neuen Merkmale werden auch in der XSD4J Bibliothek vervollständigt.

### 5.1 Einführung der XSD 1.1

XSD 1.1 basiert auf XSD 1.0 und bietet die folgenden Änderungen in der neuen Klasse [36]:

- Bugs reparieren.
- redaktionelle Verbesserungen
- konzeptionelle Klärungen und Vereinfachungen
- Alignment mit anderen Spezifikationen
- verbesserte Funktionalität und Bequemlichkeit; einschließlich der Unterstützung für eine einfachere Versionierung, Erweiterbarkeit und Abwärtskompatibilität (downward compatibility).

Die Abbildung 5.1 zeigt die Beziehung zwischen XSD 1.0 und XSD 1.1. Das bedeutet, dass XSD 1.1 eine Obermenge von XSD 1.0 ist. Es enthält mehrere leistungsstarke Funktionalitäten.

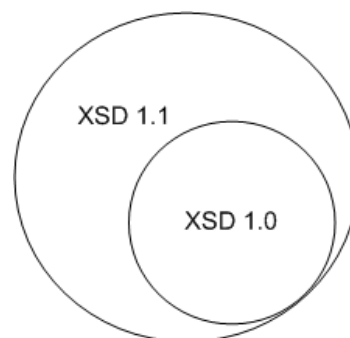


Abbildung 5.1: Beziehung zwischen XSD 1.0 und XSD 1.1

Das XSWG arbeitet derzeit an der Fertigstellung von XSD1.1. Es bietet weitgehende

<sup>1</sup>Die letzte Entwicklung basiert auf der Version W3C Working Draft 3 December 2009

<sup>2</sup>Part 1: Structures: <http://www.w3.org/TR/2009/WD-xmlschema11-1-20091203/>

<sup>3</sup>Part 2: Datatype: <http://www.w3.org/TR/2009/WD-xmlschema11-2-20091203/>

Kompatibilität mit dem XML-Schema 1.0, d.h., dass der Validator der Version 1.1 für das Instanzdokument 1.0 weiter funktioniert und der Validator der Version 1.0 nicht für das Instanzdokument 1.1 unterstützen kann. Dies entspricht den Regeln der Abwärtskompatibilität. Die Abbildung 5.2 illustriert die Beziehung des Validators zwischen XSD 1.0 und XSD 1.1.

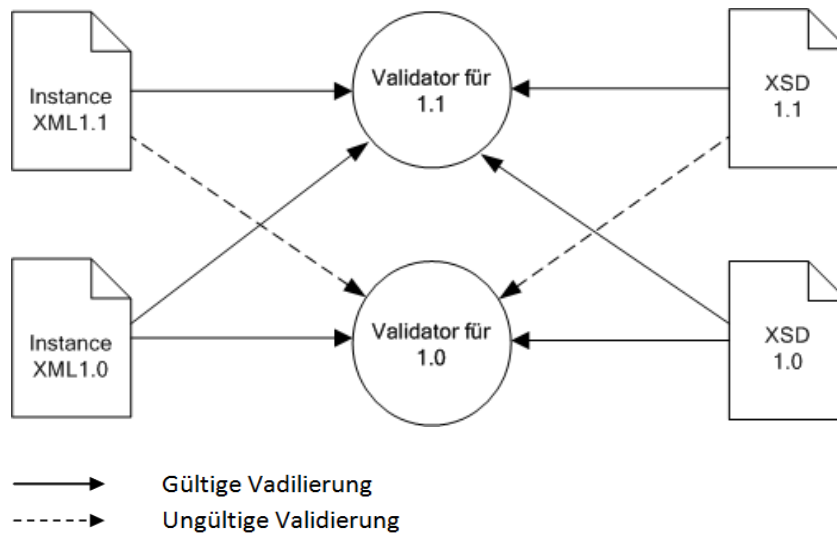


Abbildung 5.2: Beziehung des Validators zwischen XSD 1.0 und XSD 1.1

## 5.2 Die neuen Strukturen

Das konkrete neue Update hat folgende Schwerpunkte: Assertions, Open-Content, Conditional Type Assignment (CTA) und Schema-wide-Attributes. Assertions ist ähnlich wie die Überprüfung von SQL-Klauseln [36], um zusätzliche Bedingung in Form von Prädikaten (XPath-Ausdruck) abzufragen. Open-Content enthält die Elemente `<xsd:openContent>` und `<xsd:defaultOpenContent>`. Das Element `<xsd:openContent>` wird im komplexen Datentyp verwendet, dessen jeweiligen Elemente werden den Definitionsinhalt des Elements `<xsd:openContent>` einsetzen. Das Element `<xsd:defaultOpenContent>` ist ein Schema-wide-Open-Content, dieses Element soll nach den Elementen `<xsd:include>`, `<xsd:import>`, `<xsd:redefine>` und `<xsd:override>` im XSD-Dokument platziert werden. Der Definitionsinhalt des Elements `<xsd:defaultOpenContent>` kann in die jeweiligen Elemente im XSD-Dokument eingesetzt werden. CTA kann die bedingten Datentypen von einem Element auswählen, um die Attribute vererbt werden zu können. Schema-wide-Attributes `defaultAttributes` kontrolliert die Attribute, die im Element `<xsd:attributeGroup>` definiert sind, ob von jeweiligen komplexen Datentypen als Default-Attribute im XSD-Dokument referenziert sind. In dem folgende Abschnitt werden diese Strukturen von XSD 1.1 konkret angegeben.

Die Tabelle 5.1 zeigt die neuen Elemente und Attribute in XSD 1.1 und in welchen Elementen diese verwendet werden. Der Inhalt dieser Tabelle stammt aus dem Vergleich von den Strukturen der XSD 1.0 und XSD 1.1 Spezifikationen.

Die folgende Tabelle stellt die verfügbaren neuen Strukturen und Facetten in XSD 1.1 dar, die in den Elementen oder Attributen angewendet werden, in alphabetischer Reihenfolge geordnet zusammen:

Name in XSD 1.1	Angewandete Elemente
alternative	<element>
annotation	<schema>
any	<all>
assert	<b>&lt;complexType&gt;</b> , <simpleContent> <b>&lt;restriction&gt;</b> , <b>&lt;extension&gt;</b> </simpleContent>, <complexContent> <b>&lt;restriction&gt;</b> , <b>&lt;extension&gt;</b> </complexContent>, <simpleType> <b>&lt;extension&gt;</b> </simpleType>
assertion	<simpleContent> <restriction> </simpleContent>, <simpleType> <restriction> </simpleType>
defaultAttributesApply	<complexType>
defaultOpenContent	<schema>
explicitTimezone	<simpleType> <restriction> </simpleType>
inheritable	<attribute>
maxScale	<simpleContent> <restriction></simpleContent>, <simpleType> <restriction> </simpleType>
minScale	<simpleContent> <restriction></simpleContent>, <simpleType> <restriction> </simpleType>
notNamespace	<any>,<anyAttribute>
notQname	<any>,<anyAttribute>
openContent	<complexType>, <complexContent> <restriction>, <extension> </complexContent>
override	<schema>
ref	<unique>,<key>,<keyref>
targetNamespace	<attribute>,<element>,<schema >
xpathDefaultNamespace	<selector>,<field>,<schema>

Tabelle 5.1: Die Anwendungen der verfügbaren neuen Strukturen und Facetten in den Elementen von XSD 1.1

Ein Beispiel für die Verwendung der Tabelle 5.1 wird im Folgende vorgestellt:  
Das Element `<xsd:assert>` in XSD 1.1 kann im komplexen Datentyp `<xsd:complexType>`,

in den Unterelmenten `<xsd:restriction>` und `<xsd:extension>` von den Elementen `<xsd:simpleContent>` und `<xsd:complexContent>` sowie im Unterelement `<xsd:extension>` von dem Element `<xsd:simpleType>` verwendet werden. Die Fettschriften zeigen die Anwendungsbereichen des Elements `<xsd:assert>`.

Quelle: <http://www.w3.org/TR/xmlschema-1/>  
<http://www.w3.org/TR/xmlschema11-1/>

## 5.2.1 Namensräume für XSD 1.1

Die Zuordnung zum Namensraum in XSD 1.1 erfolgt durch die Deklaration:

```
xmlns:xsd="http://www.w3.org/2001/XMLSchema"
```

Das Präfix `xsd` verweist darauf, dass die verwendeten Namen zum Namensraum des XSD-Dokuments gehören. Es ist möglich, auch andere Präfixe zu verwenden. Zur Verwendung innerhalb eines Instanzdokument, das einem solchen Schema entspricht, ist noch ein weiterer Namensraum vorgegeben, der über den URI:

```
http://www.w3.org/2001/XMLSchema-instance
```

identifiziert wird, in der Regel verknüpft mit dem Präfix `xsi` [79].

In XSD 1.1 verwenden die Namensräume auch die gleiche Deklaration wie XSD 1.0. Das folgende Beispiel zeigt die Namensraumdeklaration im XML-Schema:

Listing 5.2.1: Ein XML-Fragment mit den Namensräumen

```
<?xml version="1.0" encoding="UTF-8"?>
<xsd:schema xmlns:xsd="http://www.w3.org/2001/XMLSchema"
  targetNamespace="http://www.example.org"
  xmlns="http://www.example.org"
  elementFormDefault="qualified">
</xsd:schema>
```

Aber im Gegensatz zu XSD 1.0 hat XSD 1.1 einen neuen Namensraum, der als kontrollierter Namensraum der Vision bezeichnet wird. Die Deklaration sieht wie folgt aus:

```
xmlns:vs="http://www.w3.org/2007/XMLSchema-versioning"
```

Das Präfix `vs` verweist darauf, dass sich die Nutzung dieses Namensraums hier als eine Angelegenheit der Namensraum-Policy darstellt und mehrere Namen in diesem Namensraum von dieser Spezifikation oder anderen Spezifikationen in zukünftigen Versionen angegeben werden können. Es ist möglich, auch andere Präfixe zu verwenden.

Quelle: <http://www.w3.org/TR/2009/WD-xmlschema11-1-20091203/#xsd-nss>

## 5.2.2 Schema-Namensraum-Version

Immer wenn ein konformer XSD-Prozessor ein Schema-Dokument liest, um die enthaltenen Komponenten in einem Schema zu definieren, wird zunächst über das Schema-Dokument eine Pre-Verarbeitung durchgeführt [76].

Jedes Element wird im Schema-Dokument überprüft, ob die Attribute *vc:minVersion*, *vc:maxVersion*, *vc:typeAvailable*, *vc:typeUnavailable*, *vc:facetAvailable* oder *vc:facetUnavailable* in dessen Attributen erscheint.

Das folgende Beispiel zeigt die neue Namensraumdeklaration in XSD 1.1. Dieses Beispiel findet sich in Listing 5.2.2. Das dargestellte Dokument definiert die Elemente *e1* und *e2*. In diesem Fall wird das Element *e1* durch das Attribut *vc:minVersion* eingeschränkt und seine minimale Namensraumversion ist 3.2. Das Element *e2* wird durch die Attribute *vc:minVersion* und *vc:maxVersion* eingeschränkt und seine Wertebereichen der Namensraumversion bleiben zwischen 1.1 und 3.2 (aber nicht einschließlich).

Listing 5.2.2: schema.xsd

```
<?xml version="1.0" encoding="UTF-8"?>
<xsd:schema xmlns:xsd="http://www.w3.org/2001/XMLSchema"
  xmlns:vc="http://www.w3.org/2007/XMLSchema-versioning">
  <xsd:element name="e1" vc:minVersion="3.2" />
  <xsd:element name="e2" vc:minVersion="1.1" vc:maxVersion="3.2" />
</xsd:schema>
```

Als Konvention des Namensraum-Version-Präfixes benutzen wir *vc* in der Namensraum-Deklaration. Es gibt die folgenden sechs verschiedenen Attribute. Tabelle 5.2 zeigt die Arten und die Beschreibungen der verschiedenen Schema-Version-Attribute:

Art	Beschreibung
<i>vc:minVersion</i> <i>vc:maxVersion</i>	Die Attribute <i>vc:minVersion</i> und <i>vc:maxVersion</i> können ein Element definieren, um die Version der Deklaration der Schema-Spezifikation zu beschreiben.
<i>vc:typeAvailable</i> <i>vc:typeUnavailable</i>	Die Attribute <i>vc:typeAvailable</i> und <i>vc:typeUnavailable</i> können ein Element definieren, um ein Schema zu validieren, und sie beschreiben einen eindeutigen Datentyp in diesem Element
<i>vc:facetAvailable</i> <i>vc:facetUnavailable</i>	Die Attribute <i>vc:facetAvailable</i> und <i>vc:facetUnavailable</i> können ein Element definieren, um ein Schema zu validieren, und sie beschreiben eine einzigartige Facette in diesem Element.

Tabelle 5.2: Die Attribute der Schema-Namensraum-Version

Außerdem werden mehrere Namensraumpräfixe üblicherweise im Schema-Dokument verwendet. Die folgenden Bindungen werden akzeptiert, z.B. :

- *fn* verbinden zu <http://www.w3.org/2005/xpath-functions>
- *rddl* verbinden zu <http://www.rddl.org/>
- *xlink* verbinden zu <http://www.w3.org/1999/xlink>

### 5.2.3 Deklaration von Assertions

Die Attribute können vererbbar erklärt werden. Die Elemente `<xsd:assert>` und `<xsd:alternative>` verwenden diese vererbbaren Attribute auf die Kindelemente. Der folgende Abschnitt wird die neuen Elemente `<xsd:assert>` und `<xsd:alternative>` beschrieben [44].

### 5.2.4 Element `<xsd:assert>`

Das Element `<xsd:assert>` als Assertion-Komponente kann die Existenz und die Werte der verbundenen Elemente und Attribute beschränken. Es bietet einen Mechanismus zur Überprüfung, dessen Ergebnis beurteilt wird, ob der XPath-Ausdruck (XPath 2.0) über das Inhaltsmodell eines Elements wahr ist. Dieser XPath-Ausdruck ist ein boolescher Wert. Wenn der Wert des XPath-Ausdrucks „true“ ist, wird das Element `<xsd:assert>` für den Funktionsbereich gültig sein. Dieser Funktionsbereich ist von dem Element `<xsd:assert>` definiert. Wenn der Wert des XPath-Ausdrucks „falsch“ ist, wird das Ergebnis des Elements `<xsd:assert>` als Fehlermeldung ausgegeben. Das Element `<xsd:assert>` verlangt ein Min-Attribut und ein Max-Attribut. Der Wert des Min-Attributs sollte kleiner als oder genau so groß wie der Wert des Max-Attributs sein, sonst wird diese Fehlermeldung hergestellt [48].

Listing 5.2.4 zeigt ein Beispiel des komplexen Datentyps *Tag* mit dem Element `<xsd:assert>`. Der Wert des Elements *Vormittag* darf nicht größer als der Wert des Elements *Nachmittag* sein.

Listing 5.2.4: assert.xsd

```
<xsd:complexType name="Tag" >
  <xsd:sequence>
    <xsd:element name="Vormittag" type="xsd:time" />
    <xsd:element name="Nachmittag" type="xsd:time" />
  </xsd:sequence>
  <xsd:assert test="@Vormittag le @Nachmittag" />
</xsd:complexType>
```

Quelle: <http://www.w3.org/TR/2009/WD-xmlschema11-1-20091203/#cAssertions>

### 5.2.5 Bedingter Typ `<xsd:alternative>`

Das Element `<xsd:alternative>` ist ein bedingter Typ<sup>4</sup>, d.h., es beschreibt eine Auswahl eines Datentyps. Dieser ausgewählte Datentyp basiert auf dem Ergebnis des Attributs *test*.

```
<alternative test="XPath-Ausdruck" type="type" />
```

Wenn der Wert des XPath-Ausdrucks „true“ ist, wird dieser Datentyp *type* ausgewählt. Listing 5.2.5.1 zeigt ein Beispiel des bedingten Typs `<xsd:alternative>`. Das Element

<sup>4</sup>Englisch: Conditional Type Alternatives (a.k.a. CTA)



*Kommunikation* enthält zwei alternative Datentypen: *MobileType* und *TelephoneType*. Hier gibt es die drei Möglichkeiten:

- Wenn der Wert des Attributs *kind* der Datentyp „*MobileType*“ ist, wird der Datentyp *MobileType* als Datentyp des Elements *Kommunikation* verwendet.
- Wenn der Wert des Attributs *kind* der Datentyp „*TelephoneType*“ ist, wird der Datentyp *TelephoneType* als Datentyp des Elements *Kommunikation* verwendet.
- Wenn der Wert des Attributs *kind* nicht die Datentypen *MobileType* und *TelephoneType* ist, wird der Datentyp *KommunikationType* als Datentyp des Elements *Kommunikation* verwendet.

Listing 5.2.5.1: alternative.xsd

```
<xsd:element name="Example">
  <xsd:complexType>
    <xsd:sequence>
      <xsd:element name="Kommunikation" type="KommunikationType"
                    maxOccurs="unbounded">
        <xsd:alternative test="@kind eq 'mobile'" type="MobileType" />
        <xsd:alternative test="@kind eq 'telephone'" type="TelephoneType" />
      </xsd:element>
    </xsd:sequence>
  </xsd:complexType>
</xsd:element>
```

Der Datentyp in `<xsd:alternative>` muss aus dem deklarierten Datentyp des Elements abgeleitet werden. Das zeigt die Abbildung 5.3:

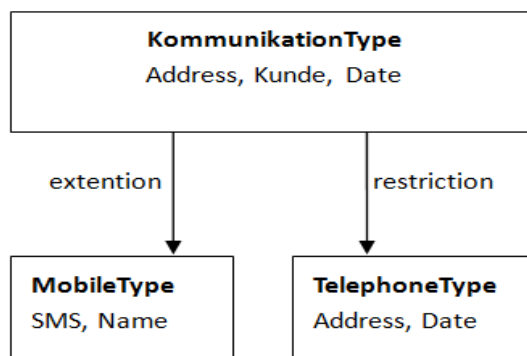


Abbildung 5.3: Die Struktur von Kommunikation

Der Datentyp *MobileType* kann in der Abbildung 5.3 durch *extention* aus dem Datentyp *KommunikationType* abgeleitet und der Datentyp *TelephoneType* kann durch *restriction* aus dem Datentyp *KommunikationType* abgeleitet werden. Listing 5.2.5.2 zeigt ausführlich die Definitionen der Datentypen *KommunikationType*, *MobileType* und *TelephoneType*. Die Datentypen *MobileType* und *TelephoneType* können innerhalb des bedingten Typs `<xsd:alternative>` durch die untergeordneten Elemente

<xsd:complexType> und <xsd:simpleType> definiert werden. Listing 5.2.5.3 zeigt die Datentypen *MobileType* und *TelephoneType* innerhalb des bedingten Typs <xsd:alternative>.

Es ist wichtig, dass der XPath-Ausdruck in dem Element <xsd:alternative> nur das Attribut des aktuellen Elements referenzieren kann. Er kann nicht die Eltern- und Kindelemente referenzieren, z.B. in Listing 5.2.5.1 referenziert der XPath-Ausdruck des Attributs *test* nur auf das Attribut *Kommunikation*.

Listing 5.2.5.2: kommunikationtype-alternative.xsd

```

<xsd:complexType name="KommunikationType">
  <xsd:sequence>
    <xsd:element name="Address" type="xsd:string"/>
    <xsd:element name="Kunde" type="xsd:string"
      minOccurs="0" maxOccurs="unbounded"/>
    <xsd:element name="Date" type="xsd:gYear"/>
  </xsd:sequence>
  <xsd:attribute name="kind">
    <xsd:simpleType>
      <xsd:restriction base="xsd:string">
        <xsd:enumeration value="mobile" />
        <xsd:enumeration value="telephone" />
      </xsd:restriction>
    </xsd:simpleType>
  </xsd:attribute>
</xsd:complexType>
<xsd:complexType name="MobileType">
  <xsd:complexContent>
    <xsd:extension base="KommunikationType">
      <xsd:sequence>
        <xsd:element name="SMS" type="xsd:string"/>
        <xsd:element name="Name" type="xsd:string"/>
      </xsd:sequence>
    </xsd:extension>
  </xsd:complexContent>
</xsd:complexType>
<xsd:complexType name="TelephoneType">
  <xsd:complexContent>
    <xsd:restriction base="KommunikationType">
      <xsd:sequence>
        <xsd:element name="Address" type="xsd:string"/>
        <xsd:element name="Date" type="xsd:gYear"/>
      </xsd:sequence>
    </xsd:restriction>
  </xsd:complexContent>
</xsd:complexType>

```

Listing 5.2.5.3: kommunikationtype-inneren-alternative.xsd

```

<xsd:element name="Kommunikation" type="KommunikationType" maxOccurs="unbounded">
  <xsd:alternative test="@kind eq 'mobile'">
    <xsd:complexType>
      ... ..
    </xsd:complexType>
  </xsd:alternative>
  <xsd:alternative test="@kind eq 'Telephone'">
    <xsd:complexType>
      ... ..
    </xsd:complexType>
  </xsd:alternative>
</xsd:element>

```

Quelle: <http://www.w3.org/TR/2009/WD-xmlschema11-1-20091203/#cTypeAlternative>

## 5.2.6 Vergleich von `<xsd:assert >` und `<xsd:alternative>`

Listing 5.2.5.2 kann durch das Element `<xsd:assert>` anstatt `<xsd:alternative>` ausgedrückt werden. Listing 5.2.6 zeigt ein XSD-Dokument. Das Element `<xsd:assert>` stellt alle Elemente vor, diese Elemente werden dann auf den XPath-Ausdruck verwendet. Dieser XPath-Ausdruck realisiert die Bedingung des Elements `<xsd:alternative>`.

Listing 5.2.6: publictiontype-assert.xsd

```

<xsd:complexType name="KommunikationType">
  <xsd:sequence>
    <xsd:element name="Address" type="xsd:string"/>
    <xsd:element name="Kunde" type="xsd:string" minOccurs="0" maxOccurs="unbounded"/>
    <xsd:element name="Date" type="xsd:gYear"/>
    <xsd:element name="SMS" type="xsd:string" minOccurs="0"/>
    <xsd:element name="Name" type="xsd:string" minOccurs="0"/>
  </xsd:sequence>
  <xsd:attribute name="kind" type="xsd:string" />
  <xsd:assert test="XPath Expression" />
</xsd:complexType>

```

Die folgenden Situationen zwischen `<xsd:assert>` und `<xsd:alternativ>` werden in der Praxis verwendet:

- Sie könnten das Element `<xsd:alternative>` auswählen, wenn Sie bequem per CTA, z.B. das Element `<xsd:alternative>`, verwenden. Ansonsten verwenden Sie `<xsd:assert>`.
- In einem Schema-Validator ist wahrscheinlicher eine Streaming-Implementierung für CTA als für die Assertion anzunehmen.

- Der Schema-Validator produziert wahrscheinlich bessere Diagnosen, wenn man die Einschränkung per CTA beschreibt.
- Die Verwendung von CTA liefert wahrscheinlich eine genauere Typ-Annotation des Elements. Dies erzielt beim Schreiben von Schema-Aware- Stylesheets und Abfragen gute Ergebnisse<sup>5</sup>.
- Die Verwendung von `<xsd:assert>` anstatt `<xsd:alternative>` hat den Vorteil, dass mehrere Informationen ausgedrückt werden können. Mit dem Element `<xsd:alternative>` kann ausschließlich eine allerdings begrenzte Basis auf die Attribute des aktuellen Elements beschrieben werden<sup>6</sup>.

## 5.2.7 Deklaration von defaultAttributes (Schema-wide-Attributes)

Default-Attribut-Gruppen werden nun unterstützt. Das Element `<xsd:schema>` kann ein Attribut `defaultAttributes` tragen, das eine benannte Definition des Attributs identifiziert. Alle komplexen Typen werden im Schema-Dokument definiert und dann werden automatisch auch die Attribut-Gruppen integriert, ohne dass der Wert des Attributs `defaultAttributesApply` im Element `<xsd:complexType>` „false“ ist.

Listing 5.2.7 zeigt die definierten Attribute in `attributeGroup`. Das Attribut `id` ist im Instanzdokument erforderlich, das Attribut `class` ist optional.

Wenn das Attribut `defaultAttributesApply` im Element `<xsd:complexType>` auftritt und dessen Wert „false“ ist, funktionieren die definierten Attribute in `attributeGroup` für diese Bedingung nicht.

```
<xsd:complexType defaultAttributesApply="false">
```

Listing 5.2.7: defaultAttributes.xsd

```
<xsd:schema xmlns:xsd="http://www.w3.org/2001/XMLSchema"
  targetNamespace="http://www.example.org"
  xmlns="http://www.example.org"
  defaultAttributes="myDefaultAttributes"
  elementFormDefault="qualified">
  <xsd:element name="Telephone">
    <xsd:complexType>
      <xsd:sequence> ... </xsd:sequence>
    </xsd:complexType>
  </xsd:element>
  <xsd:attributeGroup name="myDefaultAttributes">
    <xsd:attribute name="id" type="xsd:ID" use="required" />
    <xsd:attribute name="class" type="xsd:NMTOKENS" />
  </xsd:attributeGroup>
</xsd:schema>
```

<sup>5</sup> Michael Kay Director, Saxonica Limited, <http://saxonica.blogharbor.com/>

<sup>6</sup> Roger Costello, <http://www.xfront.com/microformats/>

### 5.2.8 <xsd:openContent> und <xsd:defaultOpenContent>

Die Elemente `<xsd:openContent>` und `<xsd:defaultOpenContent>` können die deklarierten Elemente im Instanzdokument erweitern.

Das Element `<xsd:openContent>` wird nur im komplexen Datentyp verwendet [62].

Listing 5.2.8.1 zeigt, dass beliebige Elemente, die aus einem beliebigen Namensraum kommen, das Instanzdokument einsetzen können. Dieses Instanzdokument enthält die Elemente, die vom Element `<xsd:sequence>` definiert sind.

Listing 5.2.8.1: openContent.xsd

```

<xsd:element name="Telefon">
  <xsd:complexType>
    <xsd:openContent mode="interleave">
      <xsd:any />
    </xsd:openContent>
  <xsd:sequence>
    <xsd:element name="Kunde" type="xsd:string"/>
    <xsd:element name="Adresse" type="xsd:string" />
  </xsd:sequence>
</xsd:complexType>
</xsd:element>

```

Das mögliche Instanzdokument zeigt folgendes Listing 5.2.8.2.

Listing 5.2.8.2: openContent.xsd

```

<Telefone xmlns:new="http://www.example.org" >
  <Telefon>
    <new:Hochschule> TUD</new:Hochschule>
    <Kunde>Tim </Kunde>
    <new:Stadt>Dresden</new:Stadt>
    <Adresse>Bergstr.64 </Adresse>
  </Telefon>
</Telefone>

```

Das Attribut *mode* des Elements `<xsd:openContent>` kann drei mögliche Werte annehmen: *Interleave*, *suffix* und *none*. Der Default-Wert ist *interleave*. Das bedeutet, dass die erweiterten Elemente beliebig im Instanzdokument eingesetzt werden können, wie z.B. in Listing 5.2.8.2.

Wenn der Wert des Attributs *mode* „*suffix*“ ist, bedeutet dies, dass die erweiterten Elemente nur hinter das letzte Element im Instanzdokument eingesetzt werden können. Ein mögliches Beispiel zeigt Listing 5.2.8.3.

Listing 5.2.8.3: openContent.xsd

```

<Telefone xmlns:new="http://www.example.org" >
  <Telefon>
    <Kunde>Tim </Kunde>
    <Adresse>Bergstr.64 </Adresse>
    <new:Hochschule>TUD</new:Hochschule>
    <new:Stadt>Dresden</new:Stadt>
  </Telefon>
</Telefone>

```

Der letzte Wert „*none*“ bedeutet, dass die erweiterten Elemente nicht im Instanzdokument eingesetzt werden können.

Das Element `<xsd:defaultOpenContent>` wird nur im Wurzelement `<xsd:schema>` verwendet. Listing 5.2.8.4 zeigt, dass neue Elemente, die aus jedem Namensraum kommen, vor und nach jedem Elementen im Instanzdokument eingefügt werden können.

Listing 5.2.8.4: defaultOpenContent.xsd

```

<xsd:defaultOpenContent mode="interleave">
  <xsd:any />
</xsd:defaultOpenContent>

```

Aber die Werte des Attributs *mode* im Element `<xsd:defaultOpenContent>` haben nur zwei mögliche Ausprägungen: *Interleave* und *suffix*. Ihre Bedeutung entspricht dem Attribut *mode* im Element `<xsd:openContent>`. Des Weiteren hat das Element `<xsd:defaultOpenContent>` noch ein Attribut *appliesToEmpty*, dessen Default-Wert „*false*“ ist. Es kann kontrollieren, ob die erweiterten Elemente in leere Elemente eingefügt werden können. Der Default-Wert bedeutet, dass die erweiterten Elemente in leere Elemente nicht eingefügt werden können.

Es muss beachtet werden, dass das Element `<xsd:defaultOpenContent>` an der Spitze des Schemas nach den Elementen `<xsd:include>`, `<xsd:import>`, `<xsd:redefine>` und `<xsd:override>` platziert wird.

## 5.2.9 Ungeordneter Inhalt für das Element `<xsd:all>`

Das Element `<xsd:all>` wird in XSD 1.1 erweitert, um die Elemente mit mehreren Ereignissen zu ermöglichen. Das Element `<xsd:all>` kann das Element `<xsd:any>` als Kindelement in XSD 1.1 besitzen. Diese Verwendung wird in Listing 5.2.9.1 beschrieben. Das Element *Telefon* kann einen beliebigen Wert der Adresse annehmen und eines seiner Elemente stammt aus einem beliebigen Namensraum. Diese Verwendung ist in XSD 1.0 verboten.

Listing 5.2.9.1: all-1.xsd

```

<xsd:element name="Telefon">
  <xsd:complexType>
    <xsd:all>
      <xsd:any maxOccurs="unbounded" />
      <xsd:element name="Kunde" type="xsd:string" />
      <xsd:element name="Adresse" type="xsd:string" maxOccurs="unbounded" />
      ...
    </xsd:all>
  </xsd:complexType>
</xsd:element>

```

Listing 5.2.9.2 zeigt die Verwendung der Extension des Elements `<xsd:all>`. Trotzdem kann es im komplexen Datentyp *Woche2* erweitert werden, wenn der Inhalt des Elements `<xsd:all>` im komplexen Datentyp *Woche1* nicht leer ist. Diese Verwendung ist in XSD 1.0 auch verboten.

Listing 5.2.9.2: all-2.xsd

```

<xsd:complexType name="Woche1">
  <xsd:all>
    <xsd:element name="Test" type="xsd:string" />
    ...
  </xsd:all>
</xsd:complexType>
<xsd:complexType name="Woche2">
  <xsd:complexContent>
    <xsd:extension base="Woche1" />
    <xsd:all>
      <xsd:element name="Reise" type="xsd:string" />
      ...
    </xsd:all>
  </xsd:extension>
</xsd:complexContent>
</xsd:complexType>

```

In der folgenden Tabelle sind die in XSD 1.1 verfügbaren erweiterten Strukturen nach der aktualisierten Kennzeichnung geordnet zusammengestellt:

Name	Inhalt
attribute	targetNamespace = <u>anyURI</u> inheritable = <u>boolean</u>
element	substitutionGroup = List of <u>QName</u> <sup>7</sup> targetNamespace = <u>anyURI</u> Content: ( ... ?, ( ... , <u>alternative*</u> , ... ) )
complexType	mixed = <u>boolean</u> <sup>8</sup> defaultAttributesApply = <u>boolean</u> : true Content: ( ... , ( ...   ( <u>openContent?</u> , ... ,( ..., <u>assert*</u> )))
restriction (simpleContent)	Content: ( ... ,( ... , ( ...   <u>maxScale</u>   <u>minScale</u>   ...   <u>assertion</u>   { <i>any with namespace: ##other</i> })* )?, ... , <u>assert*</u> )
extension (simpleContent)	Content: ( ... , <u>assert*</u> )
restriction, extension (complexContent)	Content: ( ... , <u>openContent?</u> , ... , <u>assert*</u> )
All	Content: ( ... , ( ...   <u>any</u> )* )
Any	notNamespace = List of ( <u>anyURI</u>   (##targetNamespace   ##local)) notQName = List of ( <u>QName</u>   (##defined   ##definedSibling))
anyAttribute	notNamespace = List of ( <u>anyURI</u>   (##targetNamespace   ##local)) notQName = List of ( <u>QName</u>   ##defined)
key, keyref, unique	ref = <u>QName</u>
field, selector	xpathDefaultNamespace = ( <u>anyURI</u>   (##defaultNamespace   ##targetNamespace   ##local))
simpleType	final = ( ...   <u>extension</u> )
restriction (simpleType)	Content: ( ... ,( ... , ( ...   <u>maxScale</u>   <u>minScale</u>   ...   <u>assertion</u>   <u>explicitTimezone</u>   { <i>any with namespace: ##other</i> })* )
schema	defaultAttributes = <u>QName</u> xpathDefaultNamespace = ( <u>anyURI</u>   (##defaultNamespace   ##targetNamespace   ##local)) : ##local Content: ((...   <u>override</u>  ...)*, ( <u>defaultOpenContent</u> , <u>annotation*</u> )?, ... )

Tabelle 5.3: Die erweiterten Strukturen in XSD 1.1

<sup>7</sup>List of QName in XSD 1.1

<sup>8</sup>Keine Default-Wert über "mixed" in XSD 1.1



In der folgenden Tabelle sind die in XSD 1.1 verfügbaren neuen Strukturen mit der Kurzreferenz zusammengestellt:

Name	Inhalt
openContent	<pre>&lt;openContent   id = <u>ID</u>   mode = (<i>none</i>   <i>interleave</i>   <i>suffix</i>) : <i>interleave</i>   {any attributes with non-schema namespace . . .}&gt;   Content: (<u>annotation?</u>, <u>any?</u>) &lt;/openContent&gt;</pre>
alternative	<pre>&lt;alternative   id = <u>ID</u>   test = <i>an XPath expression</i>   type = <u>QName</u>   xpathDefaultNamespace = (<u>anyURI</u>   (<b>##defaultNamespace</b>   <b>##targetNamespace</b>   <b>##local</b>))   {any attributes with non-schema namespace . . .}&gt;   Content: (<u>annotation?</u>, (<u>simpleType</u>   <u>complexType?</u>)) &lt;/alternative&gt;</pre>
assert	<pre>&lt;assert   id = <u>ID</u>   test = <i>an XPath expression</i>   xpathDefaultNamespace = (<u>anyURI</u>   (<b>##defaultNamespace</b>   <b>##targetNamespace</b>   <b>##local</b>))   {any attributes with non-schema namespace . . .}&gt;   Content: (<u>annotation?</u>) &lt;/assert&gt;</pre>
defaultOpenContent	<pre>&lt;defaultOpenContent   appliesToEmpty = <u>boolean</u> : <i>false</i>   id = <u>ID</u>   mode = (<i>interleave</i>   <i>suffix</i>) : <i>interleave</i>   {any attributes with non-schema namespace . . .}&gt;   Content: (<u>annotation?</u>, <u>any</u>) &lt;/defaultOpenContent&gt;</pre>
override	<pre>&lt;override   id = <u>ID</u>   schemaLocation = <u>anyURI</u>   {any attributes with non-schema namespace . . .}&gt;   Content: (<u>annotation</u>   (<u>simpleType</u>   <u>complexType</u>   <u>group</u>   <u>attributeGroup</u>   <u>element</u>   <u>attribute</u>   <u>notation</u>))* &lt;/override&gt;</pre>

Tabelle 5.4: Die neuen Strukturen in XSD 1.1

## 5.3 Die neuen Facetten

Die abgeleiteten Datentypen werden durch die Facetten im XML-Schema eingeschränkt. Um die stärkere Leistung zu erhalten, wurden die vier neuen Facetten in XSD 1.1 eingeführt: `<xsd:assertion>`, `<xsd:explicitTimezone>`, `<xsd:minScale>` und `<xsd:maxScale>`. Die Facette `<xsd:assertion>` kann einfache Datentypen durch den XPath-Ausdruck einschränken. Die Facette `<xsd:explicitTimezone>` kann nur im einfachen Datentyp die Zeitzone einschränken. Die Facetten `<xsd:minScale>` und `<xsd:maxScale>` arbeiten mit dem neuen Datentyp `<xsd:precisionDecimal>`, um die Größe des Exponenten einzuschränken, wenn der Wert in wissenschaftlicher Schreibweise verwendet wird.

### 5.3.1 Die Facette `<xsd:assertion>`

Diese Facette wird verwendet, um einen *simpleType* einzuschränken. Sie kann mit jedem Datentyp verwendet werden. Der Wert des Attributs *test* ist ein XPath-Ausdruck, dem das eingeschränkte Element genügen muss. Außerdem kann sie auch mit anderen Facetten zusammen verwendet werden, z.B. `<xsd:minInclusive>` und `<xsd:maxInclusive>`. Listing 5.3.1 zeigt das Element *zahl*, das von dem Element `<xsd:restriction>` eingeschränkt wird. Das Element `<xsd:restriction>` besteht aus drei Facetten: `<xsd:minInclusive>`, `<xsd:maxInclusive>` und `<xsd:assertion>`. *\$value* ist eine Variable, die durch 5 teilbar sein muss. Es legt den Wertebereich des Elements *zahl* { 0,5,10..100 } fest.

Listing 5.3.1: assertion.xsd

```
<xsd:element name="zahl">
  <xsd:simpleType>
    <xsd:restriction base="xsd:integer">
      <xsd:minInclusive value="0"/>
      <xsd:maxInclusive value="100"/>
      <xsd:assertion test="$value mod 5 = 0" />
    </xsd:restriction>
  </xsd:simpleType>
</xsd:element>
```

Quelle: <http://www.w3.org/TR/2009/WD-xmlschema11-2-20091203/datatypes.html#dc-assertions>

### 5.3.2 Die Facette `<xsd:explicitTimezone>`

Diese Facette erlaubt es, im Datentyp Datum festzulegen, ob die Zeitzone erforderlich ist. Diese Facette wird nur in einem einfachen Datentyp `<xsd:simpleType>` verwendet. Die Tabelle 5.1 zeigt die Anwendungen der verfügbaren neuen Strukturen und Facetten in den Elementen oder Attributen von XSD 1.1.

Die folgende Auflistung stellt ein Beispiel der Facette `<xsd:explicitTimezone>` im XSD-Dokument dar:

Listing 5.3.2: explicitTimezone.xsd

```

<xsd:simpleType>
  <xsd:restriction base="xsd:dateTime">
    <xsd:explicitTimezone value="optional" id="id_ex" fixed="true" />
  </xsd:restriction>
</xsd:simpleType>

```

Die neue Facette `<xsd:explicitTimezone>` ist eine dreiwertige Facette. Der Wert dieser Facette ist einer der Folgenden: *required*, *prohibited*, oder *optional*. Das bedeutet, dass die Funktion der Facette `<xsd:explicitTimezone>` erforderlich, verboten oder optional ist. Diese Facette ist mit den folgenden Datentypen verwendbar: `<xsd:dateTime>`, `<xsd:time>`, `<xsd:date>`, `<xsd:gYearMonth>`, `<xsd:gYear>`, `<xsd:gMonthDay>`, `<xsd:gDay>` und `<xsd:gMonth>`, damit die Zeitzone in Offset-Datum/Zeit-Datentypen verlangt oder verboten werden kann.

Es ist eine gültige Einschränkung des Werts *timezone*, dass der Wert der Facette `<xsd:explicitTimezone>` nicht geändert werden kann, außer wenn dieser Wert *optional* ist. Es ist unabhängig davon, ob der Wert des Attributs *fixed* „true“ oder „false“ ist. Das heißt, dass der Wert des Attributs *fixed* nur relevant im Zusammenhang mit der Auswahl *optional* ist.

Quelle: <http://www.w3.org/TR/2009/WD-xmlschema11-2-20091203/datatypes.html#dc-explicitTimezone>

### 5.3.3 Die Facetten `<xsd:minScale>` und `<xsd:maxScale>`

Diese beiden Facetten werden mit dem Datentyp `<xsd:precisionDecimal>` verwendet, um die Größe des Exponenten einzuschränken, wenn man den Wert in wissenschaftlicher Schreibweise ausdrückt. Siehe hierzu auch Abschnitt 5.4.2 zum Datentyp `<xsd:precisionDecimal>`.

Listing 5.3.3: maxScale.xsd

```

<xsd:simpleType name="decimal32">
  <xsd:restriction base="xsd:precisionDecimal">
    <xsd:totalDigits value="7"/>
    <xsd:maxScale value="95"/>
    <xsd:minScale value="-96"/>
  </xsd:restriction>
</xsd:simpleType>

```

Quelle:

<http://www.w3.org/TR/2009/WD-xmlschema11-2-20091203/datatypes.html#rf-maxScale>

<http://www.w3.org/TR/2009/WD-xmlschema11-2-20091203/datatypes.html#rf-minScale>

Der folgende Links stellt die Anwendung der Facetten im XSD 1.1 Dokument dar:

<http://www.w3.org/TR/2009/WD-xmlschema11-2-20091203/datatypes.html#rf-facets>

In der folgenden Tabelle sind die in XSD 1.1 verfügbaren Strukturen der neuen Facetten zusammengestellt:

Name	Inhalt
maxScale	<pre>&lt;maxScale   fixed = <u>boolean</u> : false   id = <u>ID</u>   <b>value</b> = <u>integer</u>   {any attributes with non-schema namespace . . .}&gt;   Content: (<u>annotation?</u>) &lt;/maxScale&gt;</pre>
minScale	<pre>&lt;minScale   fixed = <u>boolean</u> : false   id = <u>ID</u>   <b>value</b> = <u>integer</u>   {any attributes with non-schema namespace . . .}&gt;   Content: (<u>annotation?</u>) &lt;/minScale&gt;</pre>
assertion	<pre>&lt;assertion   id = <u>ID</u>   <b>test</b> = <i>an XPath expression</i>   xpathDefaultNamespace = (<u>anyURI</u>   (<u>##defaultNamespace</u>     <u>##targetNamespace</u>   <u>##local</u>))   {any attributes with non-schema namespace . . .}&gt;   Content: (<u>annotation?</u>) &lt;/assertion&gt;</pre>
explicitTimezone	<pre>&lt;explicitTimezone   fixed = <u>boolean</u> : false   id = <u>ID</u>   <b>value</b> = <u>NCName</u>   {any attributes with non-schema namespace . . .}&gt;   Content: (<u>annotation?</u>) &lt;/explicitTimezone&gt;</pre>

Tabelle 5.5: Die Strukturen der neuen Facetten in XSD 1.1

## 5.4 Die neuen Datentypen

Folgende fünf neue Datentypen sind in XSD 1.1 definiert [77]: anyAtomicType, precisionDecimal, dateTimeStamp (abgeleiteter Datentyp von dem Datentyp **dateTime**), yearMonthDuration (abgeleiteter Datentyp von dem Datentyp **duration**)

unddayTimeDuration (abgeleiteter Datentyp von dem Datentyp **duration**). Sie wurden als Typdefinitionen in der Klasse XSDType der XSD4J Bibliothek deklariert, um die neuen Datentypen in der XSD4J Bibliothek zu unterstützen. In dem folgenden Abschnitt werden diese Datentypen genauer erläutert [61]

Abbildung 5.4 beschreibt die Beziehungen in der eingebauten Hierarchie der Datentypen.

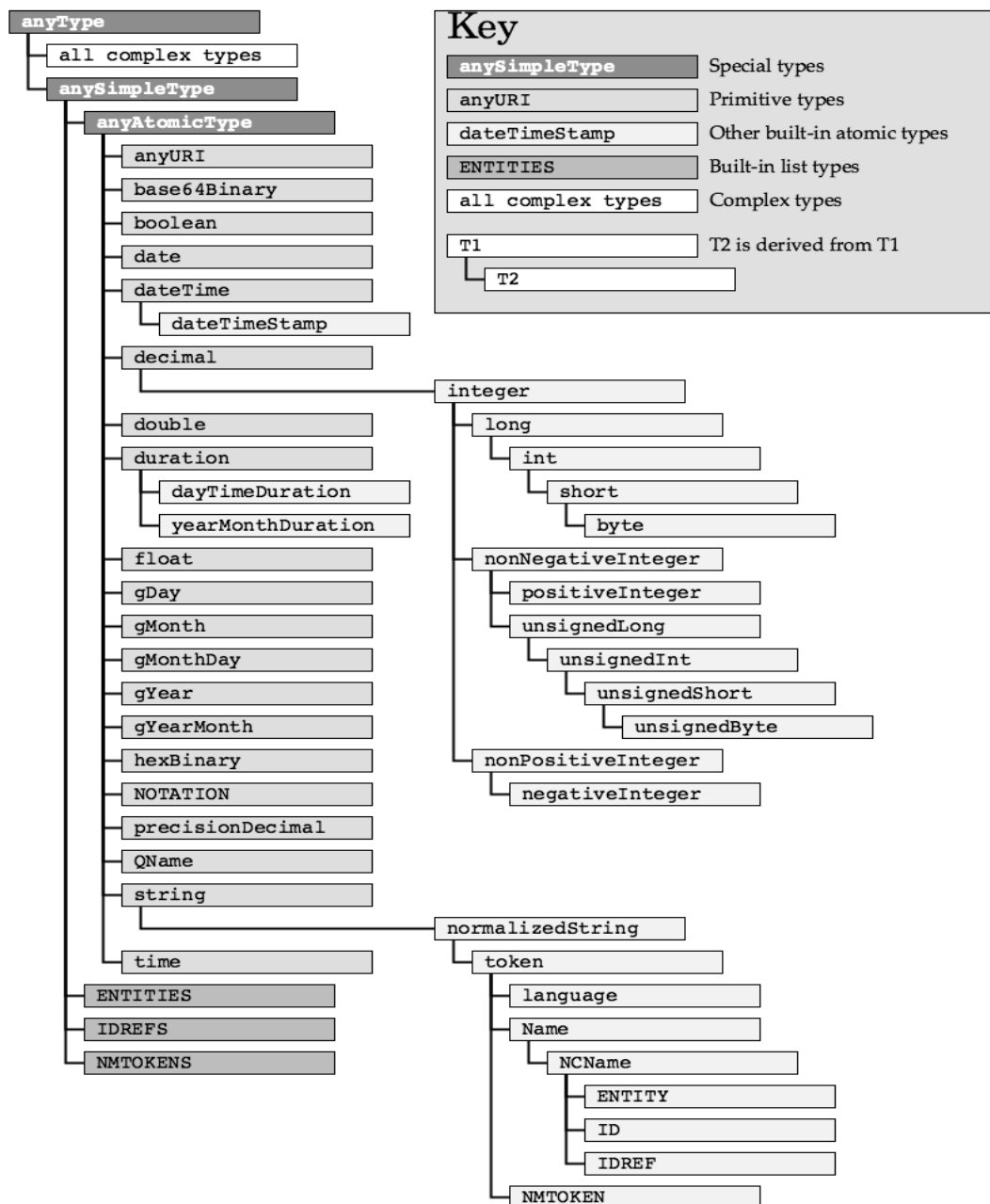


Abbildung 5.4: Datentypen der Spezifikation XSD 1.1 [37]

Die Abbildung 5.5 zeigt auf Basis der XSD 1.1 Spezifikation (Version: 03.12.2009) die abgeleitete Baumstruktur der Datentypen. Sie stellt die Veränderung der Datentypen zwischen XSD 1.0 und XSD 1.1 ausführlich dar.

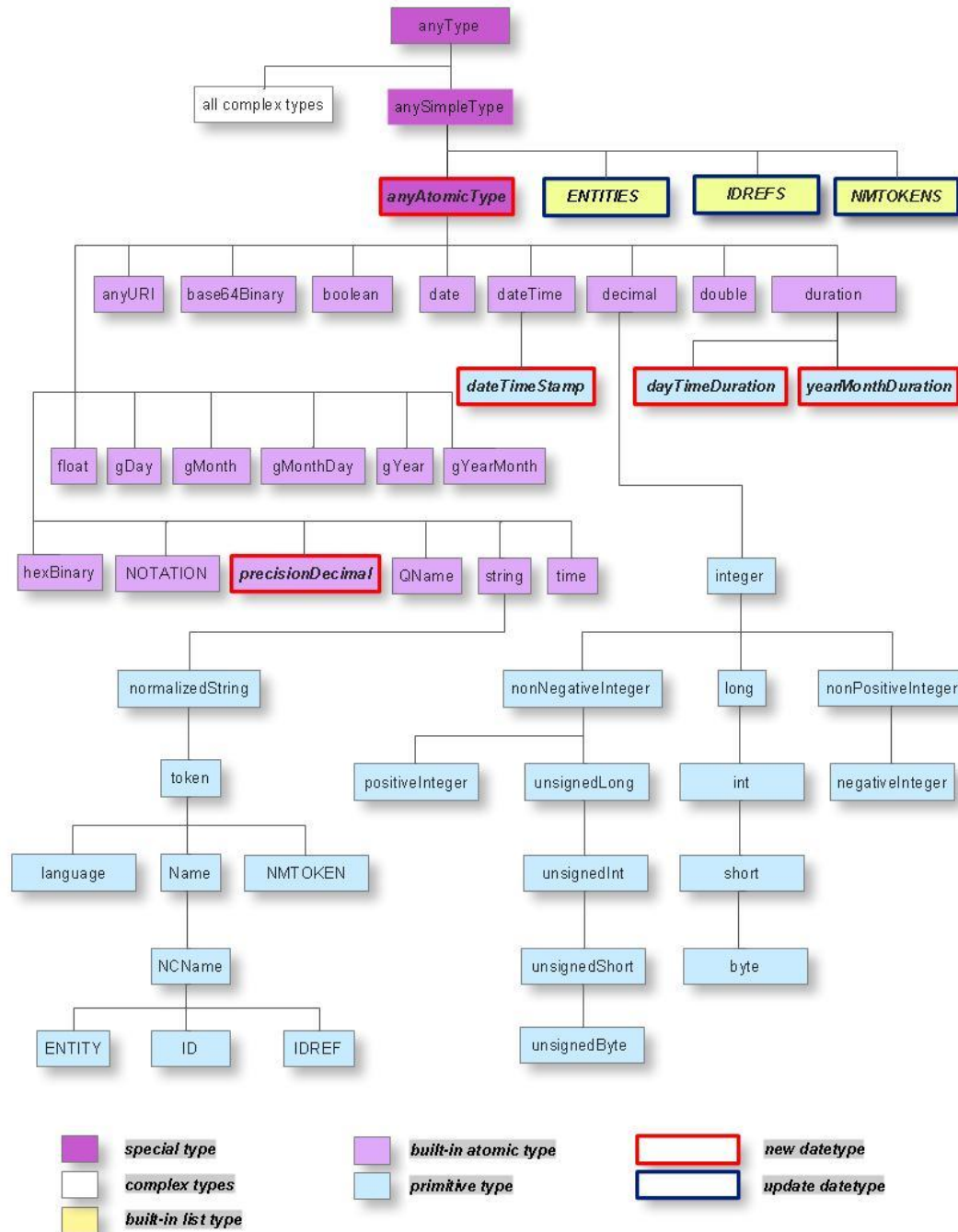


Abbildung 5.5: Die abgeleiteten Baumstrukturen der Datentypen

Quelle:

<http://www.w3.org/TR/2009/WD-xmlschema11-2-20091203/type-hierarchy-200901.longdesc.html>

### 5.4.1 Der Datentyp `<xsd:anyAtomicType>`

Der Datentyp `<xsd:anyAtomicType>` ist in einen speziellen Datentyp im XSD 1.1 eingebaut, der durch eine Einschränkung des Datentyps `<xsd:anySimpleType>` abgeleitet wird. Dieser Datentyp ist auch eine Basistypdefinition aller primitiven Datentypen. Die Basistypdefinition resultiert aus der Typdefinition, die als die Basis für die Erweiterung oder Einschränkung der Datentypen verwendet wird.

Aufgrund der W3C Empfehlung können die folgende Informationen über den Datentyp `<xsd:anyAtomicType>` festgehalten werden:

- Der Datentyp `<xsd:anyAtomicType>` ist ein neuer Datentyp.
- Die Werte und lexikalische Bereiche dieses Datentyps sind die Vereinigung der Werte und lexikalischen Bereichen aller primitiven Typen.
- Alle primitiven Typen stammen aus dem Datentyp `<xsd:anyAtomicType>`.
- Dieser Datentyp hat keine Facetten. Deshalb kann er nicht als Basistyp eines benutzerdefinierten einfachen Datentyps verwendet werden.

In der Abbildung 5.5 ist der Datentyp `<xsd:anyAtomicType>` hinter dem Datentyp `<xsd:anySimpleType>` in der Baumstruktur der Datentypen zu finden. Was ist der Unterschied zwischen dem Datentyp `<xsd:anyAtomicType>` und dem Datentyp `<xsd:anySimpleType>`? Ein Beispiel soll den Unterschied veranschaulichen:

Das Folgende zeigt die Deklarationen der Elemente über beide Datentypen im XSD-Dokument:

```
<xsd:element name="A" type="xsd:anySimpleType" />
<xsd:element name="B" type="xsd:anyAtomicType" />
```

Nachstehendes Beispiel beschreibt diese möglichen Werte von A über den Datentyp `<xsd:anySimpleType>` im XML-Dokument:

```
<A xsi:type="xsd:string">Test</A>
<A xsi:type="xsd:decimal">1.01</A>
<A xsi:type="xsd:boolean">>true</A>
<A xsi:type="alter ">32</A>
```

Hier ist *alter* als einfacher Typ definiert, in dem Facetten für die Einschränkung von numerischen Werten zur Verfügung stehen.

```
<xsd:simpleType name="alter">
  <xsd:restriction base="xsd:integer">
    <xsd:minInclusive value="18" />
    <xsd:minInclusive value="67" />
  </xsd:restriction>
</xsd:simpleType>
```

Dies sind die möglichen Werte von B über den Datentyp `<xsd:anyAtomicType>` im XML-Dokument:

```
<B xsi:type="xsd:string">Test</B>
<B xsi:type="xsd:decimal">1.01</B>
<B xsi:type="xsd:boolean">>true</B>
```

Aufgrund dieses Unterschieds zwischen beiden Datentypen in diesem Beispiel kann ein Wert in dem Facetten für den Datentyp `<xsd:anyAtomicType>` nicht angewendet werden.

Quelle: <http://www.w3.org/TR/2009/WD-xmlschema11-2-20091203/#anyAtomicType>

<http://www.w3.org/TR/2009/WD-xmlschema11-2-20091203/#anySimpleType>

### 5.4.2 Der Datentyp `<xsd:precisionDecimal>`

Mit dem Datentyp `<xsd:precisionDecimal>` wird ein neuer Datentyp im XSD 1.1 eingeführt, um den neuen IEEE-754-2008<sup>9</sup> Float Decimal Datentyp zu unterstützen. Der Datentyp `<xsd:precisionDecimal>` ist anders im Vergleich mit dem Datentyp `<xsd:Decimal>` in der Abbildung 5.5, weil er nicht nur die Präzision eines numerischen Werts ist, sondern auch die Richtigkeit eines Algorithmus behält. Der Datentyp `<xsd:precisionDecimal>` enthält auch Hinweise darauf, dass sie einen unendlichen Wert (INF), einen negativ unendlichen Wert (-INF) und einen nicht numerischen Wert (NaN) darstellen. Außerdem unterscheidet es auch zwischen dem positiven numerischen Wert 0 (+0) und dem negativen numerischen Wert 0 (-0).

Der lexikalische Raum des Datentyps `<xsd:precisionDecimal>` ist die Menge aller Dezimalzahlen (mit oder ohne Komma), der Ziffern in der wissenschaftlichen Notation (exponentielle Zahlen) und der Zeichenketten „INF“, „+INF“, „-INF“, und „NaN“.

Das folgende Beispiel zeigt eine Deklaration des Elements über den Datentyp `<xsd:precisionDecimal>` im XSD-Dokument:

```
<xsd:element name="length" type="xsd:precisionDecimal" />
```

Die folgenden Werte sind im XML-Dokument möglich:

```
<length>3.00</length> <!-- Der Wert ist 3, die Precision ist 2 -->
<length>3.0</length> <!-- Der Wert ist 3, die Precision ist 1 -->
<length>3</length> <!-- Der Wert ist 3, die Precision ist 0 -->
<length>INF</length> <!-- Der Wert ist positive unendliche Wert -->
<length>-INF</length> <!-- Der Wert ist negative unendliche Wert -->
<length>NaN</length> <!-- Der Wert ist nicht eines Number -->
```

---

<sup>9</sup>IEEE. *IEEE Standard for Floating-Point Arithmetic*. 29 August 2008.  
<http://ieeexplore.ieee.org/ISOL/standardstoc.jsp?punumber=4610933>



Durch die Einschränkung des Datentyps `<xsd:precisionDecimal>` können die benutzerdefinierten Datentypen abgeleitet werden. Diese Einschränkung besitzt die fast gleichen Eigenschaften (Facetten) wie der Datentyp `<xsd:Decimal>` und beinhaltet lediglich die Facette `fractionDigits` nicht. Darüber hinaus werden die zwei neuen Facetten `<xsd:maxScale>` und `<xsd:minScale>` eingeführt, die für die Unterstützung der abgeleiteten Datentypen verwendet werden, um die Wertebereichen des Datentyps `<xsd:precisionDecimal>` einzuschränken. Die Facette `<xsd:maxScale>` kann eine Rolle bei der Obergrenze für die Genauigkeit des Algorithmus im Datentyp `<xsd:precisionDecimal>` spielen, die Facette `<xsd:minScale>` setzt eine niedrigere Grenze in diesem Datentyp ein. Ein neuer Datentyp `Preis` wird im folgenden Beispiel definiert. Er kann die Zahlen zwischen -999,999.99 und 999,999.99 annehmen.

```
<xsd:simpleType name='price'>
  <xsd:restriction base='xsd:precisionDecimal'>
    <xsd:totalDigits value='8'/>
    <xsd:minScale value='2'/>
    <xsd:maxScale value='2'/>
  </xsd:restriction>
</xsd:simpleType>
```

In der Abbildung 5.5 ist der Datentyp `<xsd:precisionDecimal>` als neuer Datentyp in der Baumstruktur der XSD 1.1 Datentypen verwendet. Folgende Unterschiede treten zwischen den Datentypen `<xsd:precisionDecimal>` und `<xsd:Decimal>` auf:

- Der Wert eines Datentyps `<xsd:precisionDecimal>` umfasst INF, -INF und NaN, aber der Datentyp `<xsd:dezimal>` enthält unmöglich diese Werte.
- Der Wert eines Datentyps `<xsd:precisionDecimal>` kann in der wissenschaftlichen Schreibweise ausgedrückt werden, aber der Datentyp `<xsd:dezimal>` beinhaltet unmögliche Werte, die in wissenschaftlicher Notation ausgedrückt werden.
- Der Wert eines Datentyps `<xsd:precisionDecimal>` hat eine zugehörige Präzision, der Datentyp `<xsd:dezimal>` nicht.
- Der Wert eines Datentyps `<xsd:precisionDecimal>` verfügt über alle Facetten der abgeleiteten Datentypen außer der Facette `fractionDigits`, die eine Rolle für den Datentyp `<xsd:dezimal>` ähnlich wie die Facette `maxScale` für den Datentyp `<xsd:precisionDecimal>` spielt. Darüber hinaus enthält es die zwei neuen Facetten `<xsd:maxScale>` und `<xsd:minScale>`.

Quelle: <http://www.w3.org/TR/2009/WD-xmlschema11-2-20091203/#precisionDecimal>

### 5.4.3 Der Datentyp `<xsd:dateTimeStamp>`

Der Datentyp `<xsd:dateTimeStamp>` ist ein neuer Datentyp im XSD 1.1 und auch ein abgeleiteter Datentyp aus dem Datentyp `<xsd:dateTime>`. Der Datentyp `<xsd:dateTimeStamp>` ererbt alle Facetten aus dem Datentyp `<xsd:dateTime>`, aber für die Facette `<xsd:explicitTimezone>`, die als eine neue Facette im XSD 1.1 eingeführt wird,

ist der Datentyp `<xsd:dateTimeStamp>` erforderlich. Das Ergebnis aller Werte des Datentyps `<xsd:dateTimeStamp>` ist erforderlich, um explizite Zeitzone-Offsets zu erhalten. Des Weiteren ist dieser Datentyp total geordnet.

Ein Wert des Datentyps `<xsd:dateTimeStamp>` ist identisch mit dem Datentyp `<xsd:dateTime>`, außerdem erfordert er die Angabe einer Zeitzone.

Das folgende Beispiel ist eine Deklaration der Elemente über den Datentyp `<xsd:dateTimeStamp>` im XSD-Dokument:

```
<xsd:element name="birthdate" type="xsd:dateTimeStamp" />
```

Die folgenden Werte sind im XML-Dokument möglich:

```
<birthdate>1976-06-21T16:04:00-6:00</birthdate>
<birthdate>1980-01-01T24:00:00-6:00</birthdate>
```

Es gibt Unterschiede zwischen dem Datentyp `<xsd:dateTimeStamp>` und dem Datentyp `<xsd:dateTime>`: Der Datentyp `<xsd:dateTimeStamp>` ist so ähnlich wie der Datentyp `<xsd:dateTime>`, aber erfordert die Angabe einer Zeitzone. Das heißt, dass die Facette `<xsd:explicitTimezone>` für ihn eine notwendige Voraussetzung ist.

Quelle: <http://www.w3.org/TR/2009/WD-xmlschema11-2-20091203/#dateTimeStamp>

#### 5.4.4 Der Datentyp `<xsd:yearMonthDuration>`

Der Datentyp `<xsd:duration>` wird in dem Datentyp von XSD 1.0 als teilweise angeordneter Datentyp beschrieben. Er stellt einen Zeitraum dar.

Das folgende Beispiel ist eine Deklaration eines Elements über den Datentyp `<xsd:duration>` im XSD-Dokument:

```
<xsd:element name="time" type="xsd:duration" />
```

Die folgenden Werte sind im XML-Dokument möglich:

```
<time>P1M</time>
<time>P30D</time>
```

Die beide Werte des Datentyps `<xsd:duration>` können nicht verglichen werden, weil ein Monat zwischen 28 und 31 Tagen enthalten kann. Im oberen Beispiel heißt das: `P1M<>P30D`. Um die Werte vergleichbar zu machen, werden in XSD 1.1 die zwei neu total angeordneten Datentypen `<xsd:yearMonthDuration>` und `<xsd:dayTimeDuration>` eingeführt. Diese beiden Datentypen werden von den Einschränkungen des Datentyps `<xsd:duration>` abgeleitet.

Der Datentyp `<xsd:yearMonthDuration>` wird von dem Datentyp `<xsd:duration>` durch

die Einschränkung des lexikalischen Ausdrucks abgeleitet. Dieser neue Datentyp enthält nur die Komponenten Jahr und Monat.

Der lexikalische Raum des Datentyps `<xsd:duration>` ist dem Format der ISO 8601 in der Form „PnYnMnDnHnMnS“ definiert, in dem die Großbuchstaben Begrenzer sind. Wenn die entsprechenden Mitglieder nicht verwendet wird, können sie weggelassen werden.

Der Datentyp `<xsd:yearMonthDuration>` kann mit dem regulären Ausdruck:

„-?P[0-9]+(Y([0-9]+M)?|M)“

dargestellt werden. Die Werte der Komponenten Jahr und Monat ermöglichen eine vorzeichenlose Ganzzahl und das optionale Minus-Zeichen bedeutet einen negativen Datentyp `<xsd:yearMonthDuration>`.

Der Wert eines positiven Datentyps `<xsd:yearMonthDuration>` von einem Jahr und acht Monaten kann lexikalisch als P1Y8M oder P20M dargestellt werden. Diese Werte entspricht 20 Monaten.

Die folgende Werte des Datentyps `<xsd:yearMonthDuration>` sind gültig:

P1Y2M, P12Y,-P20M

Aber die folgende Werte des Datentyps `<xsd:yearMonthDuration>` sind ungültig:

P-1Y, P1Y-1M, P1YM

Das folgende Beispiel ist eine Deklaration eines Elements über den Datentyp `<xsd:yearMonthDuration>` im XSD-Dokument:

```
<xsd:element name="eventDuration" type="xsd:yearMonthDuration" />
```

Die folgenden Werte sind im XML-Dokument möglich:

```
<eventDuration>P1Y3M</eventDuration>
<eventDuration>P15M</eventDuration>
```

Die benutzerdefinierten Datentypen können durch die Einschränkung des Datentyps `<xsd:yearMonthDuration>` abgeleitet werden. Die angewendeten Facetten entsprechen so genau den Facetten des Datentyps `<xsd:duration>`. Da der Datentyp `<xsd:yearMonthDuration>` ein total angeordneter Datentyp ist, kann außerdem die Beziehung zwischen der D1 und D2 für je zwei ungleiche Werte des Datentyps `<xsd:yearMonthDuration>` hergestellt werden. Das bedeutet, dass es entweder D1>D2 oder D1<D2 gibt.

Es gibt folgende Unterschiede zwischen dem Datentyp `<xsd:yearMonthDuration>` und dem Datentyp `<xsd:duration>`:

- Der Datentyp `<xsd:yearMonthDuration>` entstammt aus der Einschränkung des Datentyps `<xsd:duration>`.
- Der Datentyp `<xsd:yearMonthDuration>` ist ein total angeordneter Datentyp, aber der Datentyp `<xsd:duration>` ist ein teilweise angeordneter Datentyp.

Quelle: <http://www.w3.org/TR/2009/WD-xmlschema11-2-20091203/#yearMonthDuration>  
<http://www.w3.org/TR/2009/WD-xmlschema11-2-20091203/#duration>

### 5.4.5 Der Datentyp `<xsd:dayTimeDuration>`

Ähnlich wie der Datentyp `<xsd:yearMonthDuration>` wird der Datentyp `<xsd:dayTimeDuration>` von dem Datentyp `<xsd:duration>` durch die Einschränkung des lexikalischen Ausdrucks abgeleitet. Es werden nur die Komponenten Tag und Uhrzeit (Stunde, Minute und Sekunde) vom Datentyp `Duration` angegeben. Der Datentyp `<xsd:yearMonthDuration>` kann mit dem regulären Ausdruck:

„`[^YM]*[DT].*`“

dargestellt werden. Die Werte der Komponenten Tage, Stunden und Minuten sind nicht beschränkt, um einen beliebigen ungekennzeichneten Datentyp `integer` zu unterstützen, und die Werte der Komponente Sekunden können einen beliebigen ungekennzeichneten Datentyp `<xsd:Decimal>` unterstützen, ebenso bedeutet das optionale Minus-Zeichen einen negativen Datentyp `<xsd:dayTimeDuration>`.

Der Wert eines positiven Datentyps `<xsd:dayTimeDuration>` von einem Tag, zwei Stunden, zwei Minuten, und 2,2 Sekunden kann lexikalisch als `P1DT2H2M2.2S` vertreten werden. Wenn die Werte von Tagen, Stunden, Minuten und Sekunden Null sind, können diese Werte weggelassen werden, weil mindestens ein Wert des Datentyps `<xsd:dayTimeDuration>` vorhanden sein muss. Der Datentyp `<xsd:dayTimeDuration>` besteht nur aus der Komponente Tage und muss das Kennzeichen „T“ weggelassen werden. Der Datentyp `<xsd:dayTimeDuration>` ist wie der Datentyp `<xsd:yearMonthDuration>` auch ein total angeordneter Datentyp.

Die folgenden Werte des Datentyps `<xsd:dayTimeDuration>` sind gültig:

`P2D`, `PT20H`, `P20DT20H`, `PT1H9M2,5S`, `-PT20M`, `-PT60.6S`

Aber die folgende Werte des Datentyps `<xsd:dayTimeDuration>` sind ungültig:

`P-5D`, `P2D3M4H5S`, `PDT12M`, `P5H`, `P2DT`

Das folgende Beispiel ist eine Deklaration eines Elements über den Datentyp `<xsd:dayTimeDuration>` im XSD-Dokument:

```
<xsd:element name="conferenceDuration" type="xsd:dayTimeDuration" />
```

Die folgenden Werte sind im XML-Dokument möglich:

```
<conferenceDuration>P35DT01H22M30S</conferenceDuration>
<conferenceDuration>PT11H</conferenceDuration>
```

Es gibt folgende Unterschiede zwischen dem Datentyp `<xsd:dayTimeDuration>` und dem Datentyp `<xsd:duration>`:

- Der Datentyp `<xsd:dayTimeDuration>` entstammt aus der Einschränkung des Datentyps `<xsd:duration>`.
- Der Datentyp `<xsd:dayTimeDuration>` ist ein total angeordneter Datentyp, aber der Datentyp `<xsd:duration>` ist ein teilweise angeordneter Datentyp.

Quelle: <http://www.w3.org/TR/2009/WD-xmlschema11-2-20091203/#dayTimeDuration>

<http://www.w3.org/TR/2009/WD-xmlschema11-2-20091203/#duration>

## 5.5 Zusammenfassung

Dieses Kapitel stellt zunächst die neuen aktuellen Eigenschaften von XSD 1.1 bezüglich der Strukturen, der Facetten und der Datentypen dar. Es handelt sich dabei um:

- Die Strukturen: Assertions, CTA, Schema-Default-Attributes, Open Content und das Element `<xsd:all>`
- Die Facetten: `<xsd:assertion>`, `<xsd:explicitTimezone>`, `<xsd:minScale>` und `<xsd:maxScale>`
- Die Datentypen: `<xsd:anyAtomicType>`, `<xsd:precisionDecimal>`, `<xsd:dateTimeStamp>`, `<xsd:yearMonthDuration>` und `<xsd:dayTimeDuration>`

Die Implementierung der Vollständigkeit der XSD4J Bibliothek basiert auf den Abläufen im Kapitel 4. Die dargestellten Eigenschaften wurden in der XSD4J Bibliothek implementiert. Die Tabelle 5.6 zeigt einen Überblick nach der vervollständigten Implementierung der XSD4J Bibliothek. Es ist möglich, dass die XSD4J Bibliothek eine kontinuierliche Aktualisierung nach der XSD-Spezifikation eingehen kann.

Arten	Namen der Testfälle und XSD Beispiel	Beschreibung
Neue Facetten	explicitTimezone explicitTimezone.xsd	Diese neue Facette wurde mit der API XSDParser und XSDDumper in XSD 1.1 Bibliothek implementiert.
	assertion assertion.xsd	Außer dem Element xpathDefaultNamespace, ist es in der Bibliothek implementiert.
	minScale minScale.xsd	Diese Facette ist in der Bibliothek implementiert.
	maxScale maxScale.xsd	Diese Facette ist in der Bibliothek implementiert.
Neue Datentypen	anyAtomicType anyAtomicType.xsd	In der XSDType ist es abgebildet, wird schon unterstützt
	dateTimeStamp dateTimeStamp.xsd	In der XSDType ist es abgebildet, wird schon unterstützt.
	dayTimeDuration dayTimeDuration.xsd	In der XSDType ist es abgebildet, wird schon unterstützt.
	yearMonthDuration yearMonthDuration.xsd	In der XSDType ist es abgebildet, wird schon unterstützt.
	precisionDecimal precisionDecimal.xsd	In der XSDType ist es abgebildet, wird schon unterstützt.
Neue Strukturen	alternative alternative.xsd	Außer dem Element xpathDefaultNamespace, ist es in der Bibliothek implementiert.

annotation annotation.xsd	Dieses Element enthält die Unterelemente <code>&lt;xsd:appinfo&gt;</code> und <code>&lt;xsd:documentation&gt;</code> . Es ist als <code>ArrayList</code> verändert, damit es einmal oder mehrmal benutzt werden kann.
any all.xsd	Das Element <code>&lt;xsd:any&gt;</code> kann im Element <code>&lt;xsd:all&gt;</code> verwendet werden.
assert assert.xsd	Dieses Element wurde in der Bibliothek unterstützt.
defaultAttributesApply defaultAttributesApply.xsd	Dieses Attribut wurde im Element <code>&lt;xsd:schema&gt;</code> unterstützt.
defaultOpenContent defaultOpenContent.xsd	Dieses Element wurde unterstützt.
inheritable inheritable.xsd	Dieses Attribut wurde im Element <code>&lt;xsd:attribute&gt;</code> unterstützt.
targetNamespace targetnamespace2.xsd	Dieses Attribut kann in den Elemente <code>&lt;xsd:schema&gt;</code> , <code>&lt;xsd:element&gt;</code> und <code>&lt;xsd:attribute&gt;</code> verwendet werden.
notNamespace	Keine Unterstützung.
notQname	Keine Unterstützung.
openContent openContent.xsd	Dieses Element kann im Element <code>&lt;xsd:complexType&gt;</code> verwendet werden.
override override.xsd	Dieses Element kann im Element <code>&lt;xsd:schema&gt;</code> verwendet werden, ist aber noch nicht vollständig.
ref	Keine Unterstützung in den Elementen <code>&lt;xsd:unique&gt;</code> , <code>&lt;xsd:key&gt;</code> und <code>&lt;xsd:keyref&gt;</code> .
xpathDefaultNamespace xpathDefaultNamespace.xsd	Dieses Attribut wird teilweise unterstützt.

Tabelle 5.6: Überblick nach der vervollständigten Implementierung von XSD4J

# Kapitel 6

## Die Testfälle von XSD

Dieses Kapitel beschreibt die Überprüfung der XSD-Dokumente in dem Test-Werkzeug XSD4JTest. Ziel ist es, herauszufinden, ob die XSD4J Bibliothek vollständig durch XSDParser realisiert ist und auch durch XSDDumper erfolgreich von dem Java-Objekt in dem genormten XSD-Dokument umgesetzt wurde. Hier wird zunächst die Überprüfung der Testfälle konzipiert, dann wird das Test-Werkzeug XSD4JTest vorgestellt. XSD4JTest kann automatisch das XSD-Dokument nach dem Parser und Dumper in der XSD4J Bibliothek überprüfen. Es kann nicht nur automatisch einen einzelnen überprüfbaren Testfall und mehrere überprüfbare Testfälle der XSD-Dokumente unterstützen, sondern auch die Testfälle der XSD-Dokumente aus der Webseite direkt durch den URL überprüfen. Des Weiteren wird die Wiedergabe kleiner grafischer Statistiken nach der Überprüfung, z.B. die Größe und Laufzeit der XSD-Dokumente, unterstützt.

### 6.1 Konzept der Testfälle

Zur Überprüfung der Testfälle wird der Tomcat Container verwendet. Die Abbildung 6.1 zeigt den Ablauf der XSD-Dokumente.

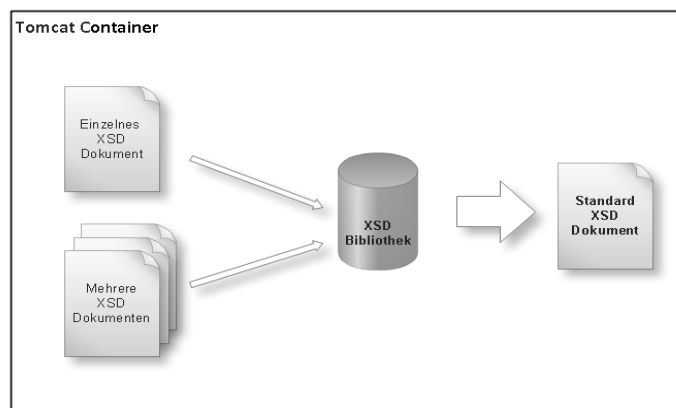


Abbildung 6.1: Konzept der Testfälle

Hier ergeben sich zwei Fälle:

#### 1. Einzelnes XSD Dokument

Um den Aufbau der XSD4J Bibliothek aufzubauen, werden zunächst das einzelne Element, Attribut und den einzelnen Datentyp usw. unterstützt. Die einzelnen Komponenten werden zunächst in der XSD4J Bibliothek überprüft. Die Realisierung der einzelnen Komponenten erfolgt nach der XSD-Spezifikation. Diese Spezifikation ist die gesetzliche Grundlage, die im Kapitel 3 (XSD 1.0) und im Kapitel 5 (XSD 1.1)

vorgestellt wird. Die einzelnen Testfälle behandeln die Überprüfung des Einzeldatentyps oder der Einzelstruktur der Elemente und Attribute.

## 2. Mehrere XSD-Dokumente

Das einzelne XSD-Dokument ist funktionsfähig. Dies bedeutet nicht, dass es im komplexen XSD-Dokument verwendbar ist. Es muss noch mit den komplexen XSD-Dokumenten überprüft werden. Diese Überprüfung kann nicht nur die Verständigkeit des XSD-Dokuments, sondern auch die Kompatibilität der verschiedenen Versionen der XSD-Spezifikation untersuchen.

## 6.2 Implementierung der XSD4JTest

XSD4JTest ist ein Test-Werkzeug für Testfälle des XSD-Dokuments, es arbeitet im Tomcat Container. Dieser Abschnitt wird die Grundlagen der Entwicklung und die Architektur von XSD4JTest vorstellen.

### 6.2.1 Das Model-View-Controller-Entwurfs-Muster (MVC)

Die Entwicklung von XSD4JTest ist abhängig von MVC [38]. Das MVC löst die Geschäftslogik aus dem Servlet heraus und verlegt sie in ein „Model“, das eine ganz normale, wiederverwendbare Java-Klasse ist. Das Modell ist eine Kombination aus den Anwendungs- oder Geschäftsdaten und den Methoden, die auf diesen Daten arbeiten. Der Controller nimmt Benutzereingaben aus einer Anfrage entgegen und findet heraus, was sie innerhalb des Modells bedeuten. Dieser Controller weist das Modell an, sich zu aktualisieren, und stellt dem View diesen neuen Zustand zur Verfügung. Ein „View“ ist verantwortlich für die Präsentation und erhält den Zustand des Modells vom Controller [53].

Die Abbildung 6.2 zeigt das MVC-Entwurfs-Muster, XSD4JTest wird mit diesem MVC-Architektur-Muster entwickelt.

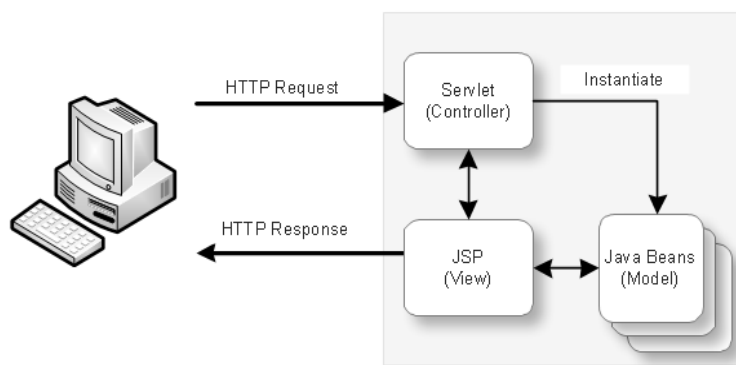


Abbildung 6.2: MVC Entwurfs Muster in XSD4JTest

Der Benutzer fordert durch den Browser die JSP-Seite an, die mit den verschiedenen XSD Dokumenten (einzelnes XSD-Dokument oder mehrere XSD-Dokumente) angefragt wird.



Der Browser sendet die Anfragedaten mit HTTP Request an den Container, dann findet der Container das richtige Servlet (XSD4JTestServlet oder anderes) anhand dem URL und leitet die Anfrage weiter. Das Servlet konsultiert die XSD4J Bibliothek und diese Bibliothek gibt eine Antwort zurück, die vom Servlet nach dem Anfrageobjekt hinzugefügt wird. Dieses Servlet leitet die Anfrage an die JSP-Seite weiter. Diese JSP-Seite nimmt die Rückgabe des Anfrageobjekts entgegen und erzeugt für den Container eine Seite. Zum Schluss gibt der Container die Seite an den Benutzer mit HTTP Response zurück [39]. Abbildung 6.3 zeigt die Verwendung der Komponenten von MVC-Entwurfs-Muster in XSD4JTest. XSD4JTestServlet ist ein Controller, die XSD4J Bibliothek spielt eine Rolle als Modell und JSPs gibt die Ergebnisse dem Benutzer als View wieder.

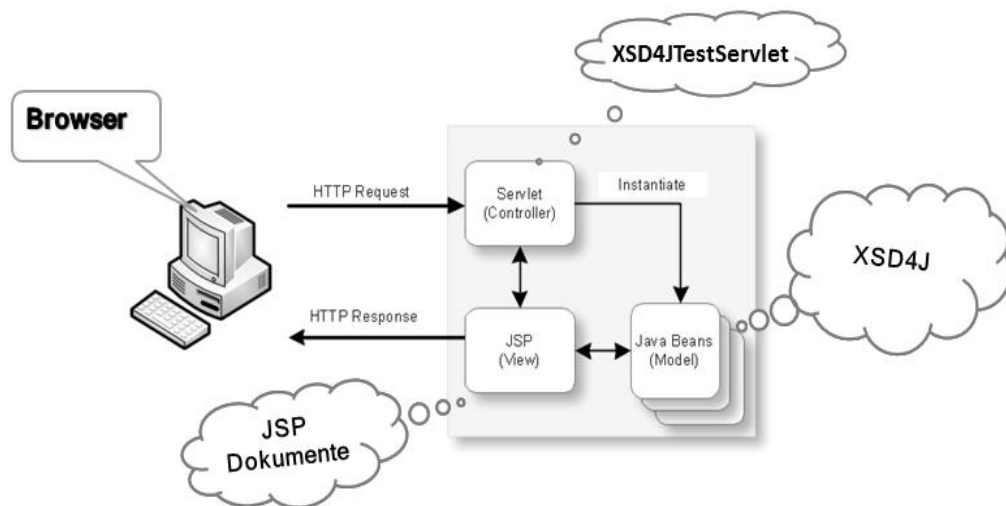


Abbildung 6.3: Die Komponenten von MVC Entwurfs Muster in XSD4JTest

## 6.2.2 Architektur von XSD4JTest

Die Abbildung 6.4 zeigt die innere Architektur von XSD4JTest. Er enthält fünf Servlets. XSD4JTestsServlet und XSD4JTestServlet können die einzelne Testfälle überprüfen. Sie können zunächst das originale XSD-Dokument mit einem Applet exportieren und dann dieses XSD-Dokument weiter bearbeiten [71]. XSD4JServlet1 und XSD4JServlet2 können mehrere Testfälle überprüfen. Sie sind abhängig von der Methode XSD4JTestChart, die die Statistiken des überprüften XSD-Dokuments mit der API von JfreeChart 1.0.13<sup>1</sup> erstellen kann. Außerdem können sie die verschiedenen XSD-Versionen verarbeiten. XSDDownloadServlet kann das XSD-Dokument überprüfen, das direkt aus der Webseite verarbeitet wird. Wenn XSD4JTest das XSD-Dokument verarbeitet, kann die XSD4J Bibliothek ein entsprechendes BootstrappedSchema (BootstappedSchema10 oder

<sup>1</sup><http://www.jfree.org/jfreechart>

BootstrappedSchema11) auswählen. BootstappendSchema10 bedient XSD 1.0. BootstrappedSchema11 bedient XSD 1.1.

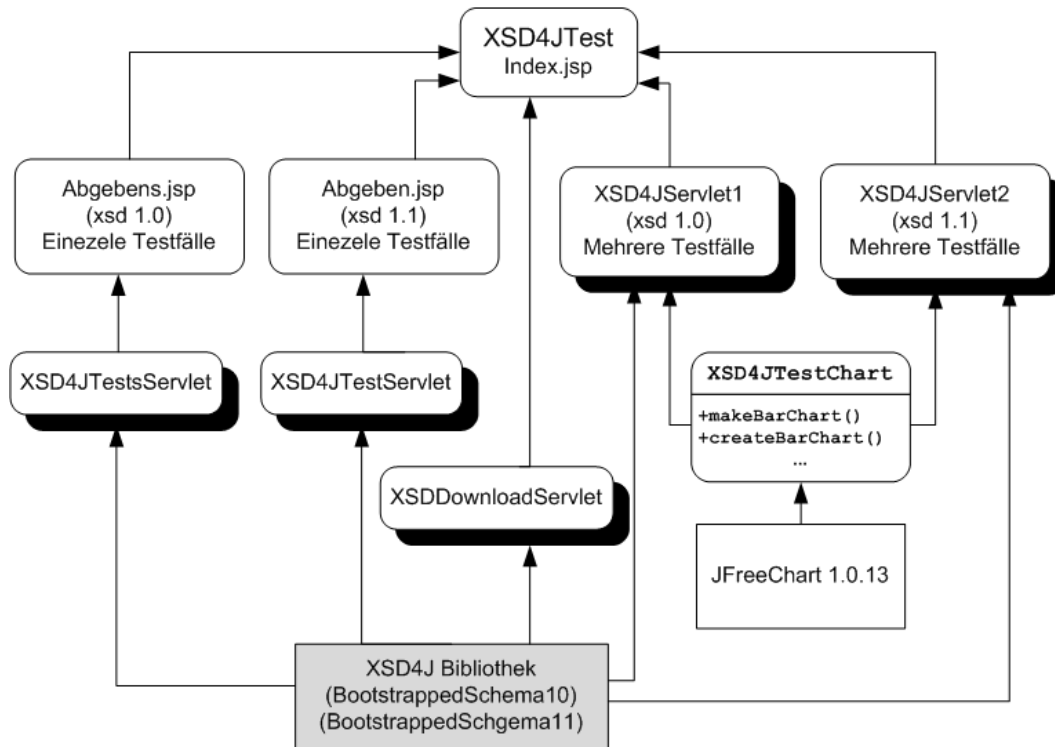


Abbildung 6.4: Die Struktur von XSD4JTest

### 6.3 Definitionen der Struktur des Dokuments (Testfälle)

Zur Überprüfung werden die verschiedenen XSD-Dokumente verwendet. Die Tabelle 6.1 spezifiziert den Namen, die XSD-Version, die Herkunft, die Umfang-Version und die Menge der überprüften XSD-Dokumente. Die URLs der Testfälle zeigt Table 6.2.

Name	XSD-Version	Herkunft	Menge	Umfang-Version
Test10	XSD 1.0	XSD4J	30	SVN-Version (Standard: 1426)
Test1_w3c	XSD 1.0, XSD 1.1	W3C, w3schools	6	Spezifikationen XSD 1.0 und XSD 1.1 (Draft)
Test2_11	XSD 1.1	W3C, w3schools	31	Spezifikationen XSD 1.0 und XSD 1.1 (Draft)
Test3_xerces_j	XSD 1.1	Xerces	26	Xerces 2_10_0 J (SVN Version 926149)
Test4_saxon	XSD 1.1	Saxon EE	8	Saxon EE 9.2.0.2 Java

Tabelle 6.1: Die Information der Testfälle

Name	http
Test10	<a href="http://dynvocation.selfip.net/download/">http://dynvocation.selfip.net/download/</a>
Test1_w3c	<a href="http://www.w3.org/XML/Schema">http://www.w3.org/XML/Schema</a> <a href="http://www.w3.org/TR/xmlschema11-1/">http://www.w3.org/TR/xmlschema11-1/</a> <a href="http://www.w3.org/TR/xmlschema11-2/">http://www.w3.org/TR/xmlschema11-2/</a> <a href="http://www.w3schools.com/schema/">http://www.w3schools.com/schema/</a>
Test2_11	<a href="http://www.w3.org/TR/xmlschema11-1/">http://www.w3.org/TR/xmlschema11-1/</a> <a href="http://www.w3.org/TR/xmlschema11-2/">http://www.w3.org/TR/xmlschema11-2/</a> <a href="http://www.w3schools.com/schema/">http://www.w3schools.com/schema/</a>
Test3_xerces_j	<a href="https://svn.apache.org/repos/asf/xerces/java/branches/xml-schema-1.1-dev/">https://svn.apache.org/repos/asf/xerces/java/branches/xml-schema-1.1-dev/</a>
Test4_saxon	<a href="http://saxon.sourceforge.net/">http://saxon.sourceforge.net/</a>

Tabelle 6.2: Die URLs der Testfälle

## 6.4 Einzelner Testfall und mehrere Testfälle

Ein einzelner Testfall bedeutet, dass der Testfall separat in der XSD4J Bibliothek überprüft wird und mehrere Testfälle bedeuten, dass die Testfälle zusammen in der XSD4J Bibliothek überprüft werden. Die folgende Überprüfungen werden getrennt nach einzelnen und mehreren Testfällen analysiert. Bei diesen Gruppen handelt es sich um die Testfälle der vervollständigten Implementierung in der Arbeit, W3C, Apache Xerces-J und Saxon EE.

### 6.4.1 Testfälle für XSD 1.1 (Implementierung in der Arbeit)

Jede einzelne und mehrere Testfälle werden 100 mal überprüft. Die Laufzeit stammt aus dem Durchschnitt der gesamten Laufzeit.

Die Tabelle 6.3 stellt das Ergebnis der Überprüfung der einzelnen und mehrerer Testfälle der vervollständigten Implementierung in der Arbeit dar. Diese Tabelle beinhaltet den Namen, die Größe und die Laufzeit des überprüften XSD-Dokuments. Hier kann man auch sehen, dass die Laufzeit eines Einzelfalls immer schneller als bei mehreren Testfällen ist. Wie das vierte Kapitel (der Abschnitt: XSD4J mit DOM) bereits gezeigt hat, basiert das XML-Dokument auf der Baumstruktur. Je größer das XSD-Dokument ist, desto langsamer wird seine relative Bearbeitungszeit sein.

Weil hier das XSD-Dokument nicht groß ist, kann man die Beziehung mit den Größen des einzelnen XSD-Dokuments nicht einfach vergleichen. Bei der Überprüfung der Testfälle von W3C wird diese Beziehung analysiert.

XSD Name	Größe (Bytes)	Laufzeit (ms) (mehrere Testfälle)	Laufzeit (ms) (einzelne Testfall)
all.xsd	920	59,32	45,34
alternative.xsd	3805	120,91	87,51
annotation.xsd	820	76,55	44,42
antribute.xsd	451	43,01	41,72
any.xsd	454	64,36	36,63
anyAtomicType.xsd	560	67,29	38,88
assert.xsd	1764	77,93	58,09
assertion.xsd	478	51,18	40,67
Book.xsd	642	48,8	37,78
BookStore.xsd	1436	76,92	75,70
dateTimeStamp.xsd	2841	92,23	69,16
dayTimeDuration.xsd	907	62,18	43,90
defaultAttributes.xsd	1429	90,54	56,65
defaultAttributesApply.xsd	1565	72,62	58,21
defaultOpenContent.xsd	1819	94,19	65,61
explicitTimezone.xsd	362	74,3	36,86
inheritable.xsd	447	42,76	40,53
maxScale.xsd	620	53,13	36,47
minScale.xsd	624	50,38	36,49
mixed.xsd	1848	50,6	40,68
officecalendar.xsd	461	65,84	40,71
openContent.xsd	628	55,0	42,55
precisionDecimal.xsd	1315	82,89	56,13
record.xsd	526	46,14	39,72
record2.xsd	572	55,26	38,61
record3.xsd	612	58,05	42,35
substitutionGroup.xsd	598	52,88	40,14
testnamespace.xsd	1429	83,02	58,88
testnamespace2.xsd	753	64,31	43,33
xpathDefaultNamespace.xsd	478	57,27	38,13
yearMonthDuration.xsd	821	63,55	44,90

Tabelle 6.3: Die Ergebnisse der Testfälle in der Implementierung

#### 6.4.2 Testfälle für XSD 1.0 und XSD 1.1 (W3C)

In diesem Abschnitt werden die größeren Daten in der XSD4J Bibliothek überprüft. Diese Daten entstammen den standardisierten Strukturen-XSD-Dokumenten und den Datentypen-XSD-Dokumenten von W3C.

Die Abbildung 6.5 zeigt das Ergebnis der überprüften Testfälle. Die Y-Achse bezieht sich auf die im XSD4JTest ausgeführte Laufzeit. Die X-Achse beschreibt den Namen des XSD-Dokuments. Zunächst können alle Testfälle im XSD4JTest ausgeführt werden. Das Struktur-XSD-Dokument von XSD 1.1 (*xsd-structure-1-1.xsd*: 969ms) benötigt eine längere Laufzeit als das Struktur-Dokument von XSD 1.0 (*xsd-structure.xsd*: 798ms). Die Datentypen-XSD-Dokumente von XSD 1.1 bestehen aus den drei XSD-Dokumenten *xsd-types-1-1-derived.xsd*, *xsd-types-1-1-primitive.xsd* und *xsd-types-1-1.xsd*. Ihre gesamte Laufzeit ( $271+624+493=1338$  ms) ist auch größer als das Datentypen-XSD-Dokument von XSD 1.0 (*xsd-types.xsd*: 1239 ms). Das bedeutet, dass die Strukturen und die Datentypen von XSD 1.1 mehr als von XSD 1.0 verwendet werden.

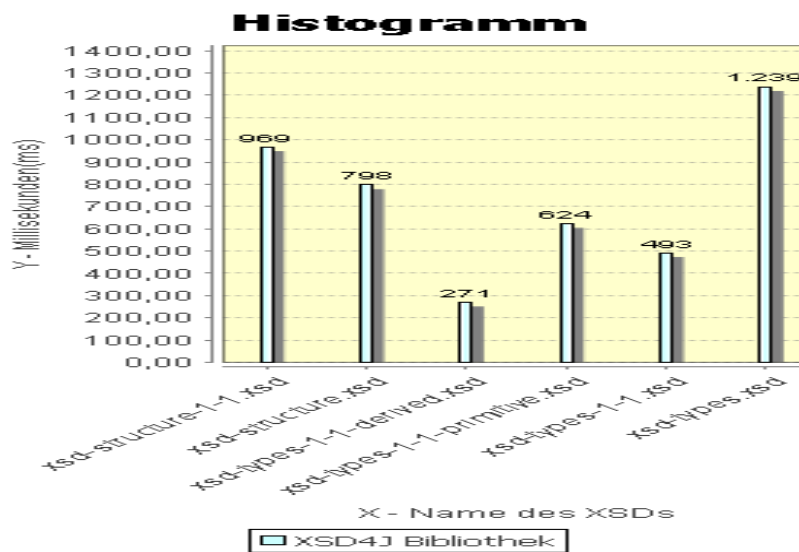


Abbildung 6.5: Gruppe Test von W3C

Die Informationen über den Einzelfall und die verschiedenen Testfälle von W3C werden in der Tabelle 6.4 dargestellt.

XSD Name	Größe (Bytes)	Laufzeit(ms) (mehrere Testfälle)	Laufzeit(ms) (einzelne Testfall)
xsd-structure-1-1.xsd	54109	969,0	893,0
xsd-structure.xsd	45656	798,0	749,0
xsd-types-1-1-derived.xsd	13703	271,0	234,0
xsd-types-1-1-primitive.xsd	21783	624,0	458,0
xsd-types-1-1.xsd	15576	493,0	287,0
xsd-types.xsd	42923	1239,0	1138,0

Tabelle 6.4: Die Ergebnisse der Testfälle von W3C

Hier kann man auch deutlich sehen, dass die Laufzeit des Einzelfalls schneller als die mehrerer Testfälle ist. Auf der anderen Seite geht es auch um die Komplexität [68] des XML-Dokuments. Wir wissen, dass die Komplexität der Datentypen höher als die der Struktur im XML-Dokument ist. In dieser Tabelle ist das Struktur-XSD-Dokument *xsd-struktur-1-1.xsd* größer als das Datentyp-XSD-Dokument *xsd-types.xsd*, aber seine Laufzeit ist kürzer als die des Datentyp-XSD-Dokuments *xsd-types.xsd*. Das bedeutet, je höher die Komplexität des XSD-Dokuments ist, desto länger wird es von der XSD4J Bibliothek bearbeitet.

### 6.4.3 Testfälle von XSD 1.1 (Apache Xerces2 Java)

Hier werden die Testfälle von Apache-Xerces2-Java in der XSD4J Bibliothek überprüft. Die Tabelle 6.5 zeigt die Ergebnisse der überprüften Testfälle. Die grafische Darstellung von mehreren Testfällen befindet sich in Abbildung C5 im Anhang. Diese Tabelle lässt folgenden Schluss zu: Die Laufzeit des Einzelfalls ist schneller als mehrerer Testfälle.

XSD Name	Größe (Bytes)	Laufzeit(ms) (mehrere Testfälle)	Laufzeit(ms) (einzelne Testfall)
a.xsd	163	42,0	27,0
address.xsd	1692	31,0	46,0
base.xsd	2412	112,0	43,0
idc.xsd	1200	69,0	30,0
ipo.xsd	1881	65,0	59,0
otherNamespace.xsd	489	39,0	26,0
personal.xsd	2218	82,0	66,0
schema_imp.xsd	409	51,0	41,0
tests.xsd	1751	62,0	39,0
xmlbase.xsd	410	63,0	44,0
XSAAttributeAnnotationsTest01.xsd	441	66,0	32,0
XSAAttributeAnnotationsTest02.xsd	1169	62,0	39,0
XSAAttributeAnnotationsTest03.xsd	569	32,0	29,0
XSAAttributeGroupAnnotationsTest01.xsd	447	44,0	34,0
XSAAttributeGroupAnnotationsTest02.xsd	792	47,0	31,0
XSAAttributeGroupAnnotationsTest03.xsd	572	69,0	23,0
XSAAttributeGroupAnnotationsTest04.xsd	1407	49,0	34,0
XSAAttributeUseAnnotationsTest01.xsd	765	65,0	24,0
XSAAttributeUseAnnotationsTest02.xsd	1593	71,0	66,0
XSAAttributeUseAnnotationsTest03.xsd	1596	56,0	49,0
XSElementTest01.xsd	1016	70,0	46,0
XSMoelGroupDefinitionTest01.xsd	587	67,0	30,0
XSMoelGroupTest01.xsd	2459	48,0	45,0

XSNotationAnnotationsTest01.xsd	986	57,0	25,0
XSParticleTest01.xsd	3970	101,0	61,0
XSWildcardTest01.xsd	1894	47,0	36,0

Tabelle 6.5: Die Ergebnisse der Testfälle von Xerces-J

#### 6.4.4 Testfälle von XSD 1.1 (Saxon EE)

Hier werden die Testfälle von Saxon EE in der XSD4J Bibliothek überprüft. Die grafischen Darstellungen von mehreren Testfällen bietet Abbildung 6.6 und 6.7.

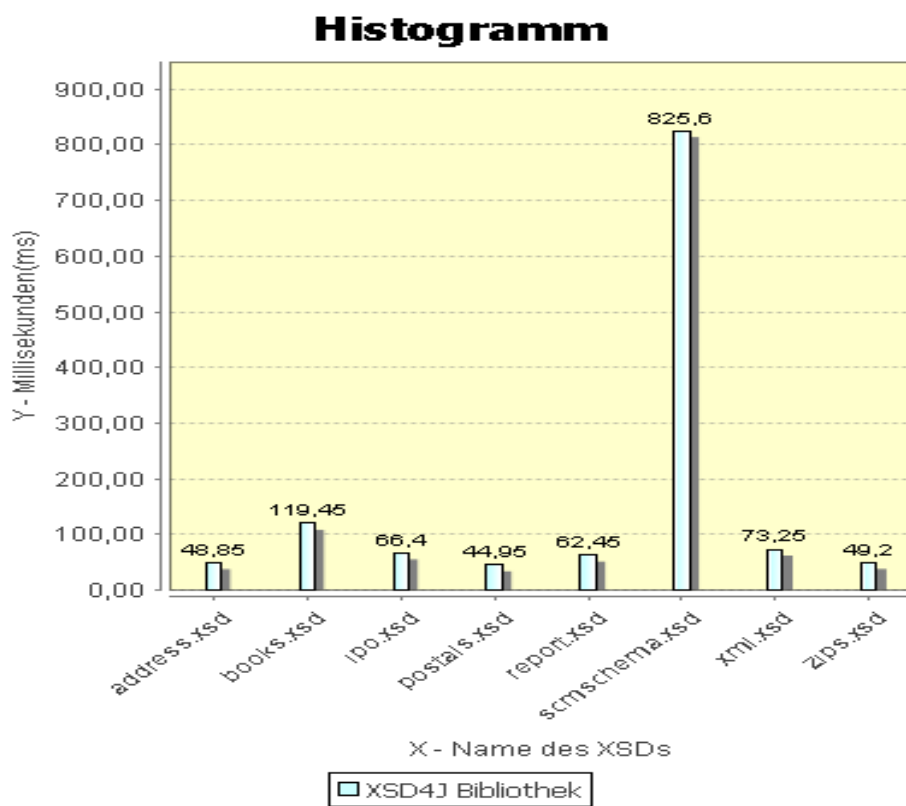


Abbildung 6.6: Gruppe Test von Saxon EE (20 mal)

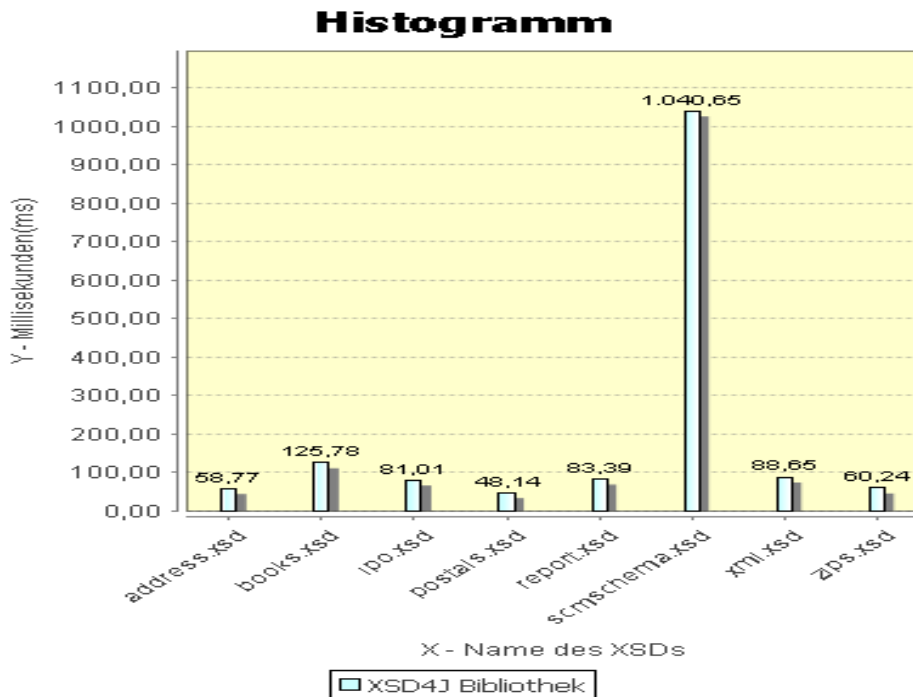


Abbildung 6.7: Gruppe Test von Saxon EE(100 mal)

Die Tabelle 6.6 zeigt die Ergebnisse der überprüften Testfälle mit der verschiedenen Frequenz. Aus dieser Tabelle ist abzulesen, dass die Laufzeit des Einzelfalls kürzer als die mehrerer Testfälle ist. Die Testfälle beinhalten die originalen XSD-Dokumente *books.xsd* und *xml.xsd*. Das XSD-Dokument *books.xsd* enthält mehr abgeleitete Datentypen als das XSD-Dokument *xml.xsd*. Daher ist festzustellen, dass die Komplexität des XSD-Dokument *books.xsd* höher als die des XSD-Dokuments *xml.xsd* ist. In der Tabelle 6.6 ist das XSD-Dokument *books.xsd* kleiner als das XSD-Dokument *xml.xsd* (4014 < 5704 Bytes), aber es ist schneller als das XSD-Dokument *xml.xsd* gelaufen (119,45 > 73,25 ms, 125,78 > 88,65 ms und 93 > 63,0 ms).

XSD Name	Größe (Bytes)	Laufzeit (ms) (mehrere Testfälle) 20mal	Laufzeit(ms) (mehrere Testfälle ) 100mal	Laufzeit (ms) (einzelne Testfall) 1mal
address.xsd	1733	48,85	58,77	34,0
books.xsd	4014	119,45	125,78	93,0
ipo.xsd	2113	66,4	81,01	50,0
postals.xsd	788	44,95	48,14	35,0
report.xsd	2167	62,45	83,39	50,0
scmschema.xsd	48543	825,6	1040,65	816,0
xml.xsd	5704	73,25	88,65	63,0
zips.xsd	869	49,2	60,24	47,0

Tabelle 6.6: Die Ergebnisse der Testfälle von Saxon EE



## 6.5 Vorstellung der APIs, Servlets und JSPs

Das Testwerkzeug XSD4JTest startet in Tomcat 6.0. Es besteht aus Java-Package, Servlets und JSPs. Nachfolger Abschnitt wird diese genauer vorstellen.

Tabelle 6.7 zeigt die APIs in Package *xsd4japp* im Test-Werkzeug XSD4JTest.

Package oder Methode	Beschreibung
xsd4japp	Dieses <i>package xsd4japp</i> enthält Servlets und eine Methode XSD4JTestChart. Die Servlets werden in der folgenden Tabelle 6.9 beschrieben.
XSD4JTestChart	Diese Klasse erstellt die Tabelle und Grafikdarstellung des XSD-Dokuments, sie enthält die Größe und Laufzeit des XSD-Dokuments. Diese Klasse ruft die APIs von <i>jfreechart-1.0.13</i> auf.

Tabelle 6.7: APIs in Package *xsd4japp*

Tabelle 6.8 zeigt die APIs in Package *applets* im Test-Werkzeug XSD4JTest.

Package oder Methode	Beschreibung
applets	Das Package <i>apples</i> enthält Applets und eine Methode <i>test</i> .
Applet <i>ResourceTest</i>	<i>public class ResourceTest extends JApplet</i> Dieses Applet kann die Inhalte von XSD lesen und durch Browser repräsentieren.
Methode <i>test</i>	<i>public test(boolean debugging, String tests)</i> Die Methode <i>test</i> entscheidet, ob das XSD-Dokument die Aktionen <i>parser</i> , <i>dumper</i> , <i>validator</i> , <i>transformer</i> und <i>instantiator</i> ausführt.
Methode <i>testrun</i>	<i>public void testrun(String xsdfile, String bootstrap, HttpServletResponse response)</i> Die Methode <i>testrun</i> ruft das XSD-Dokument auf, gleichzeitig entscheidet es, ob die Aktion <i>dumper</i> gestartet wird.
Methode <i>testparser</i>	<i>public boolean testparser(String xsdfile, String bootstrap, HttpServletResponse response)</i> Die Methode <i>testparser</i> macht den Parser des XSD-Dokuments und entscheidet, welches BootstrappedSchema gewählt wird. Am Ende wird das Ergebnis von Parser nach Servlet gesendet.
Methode <i>stats</i>	<i>private void stats(XSDSchema xsdschema, HttpServletResponse response)</i> Berechnen die Zahlen von <i>type</i> , <i>group</i> , <i>element</i> , <i>attribute</i> , <i>include</i> , und <i>import</i> im XSD-Dokument nach dem Parser.
Methode <i>testdumper</i>	<i>private boolean testdumper(String bootstrap, XSDSchema xsdschema, HashMap namespaces, HttpServletResponse response)</i> Wenn die Aktion XSDDumper nach dem Parser ausgeführt wird, wird diese Methode aufgerufen

Methode fixHTML	<i>public static String fixHTML(String rString)</i> Transformieren der besonderen Zeichen: „“ „“ „&“, „<“ und „>“ zur Erkennung des Browsers.
--------------------	--

Tabelle 6.8: APIs in Package Applet

Tabelle 6.9 zeigt die Information der Servlets im Test-Werkzeug XSD4JTest.

Package oder Methode	Beschreibung
<i>xsd4japp</i>	Das Package <i>xsd4japp</i> enthält die Servlets.
<i>XSD4JServlet1</i>	Wenn der Parser das Schema <i>BootstrappedSchema10</i> wählt, ruft dieses Servlet das XSD-Dokument auf, und überprüft alle XSD-Dokumente zusammen. Außerdem speichert es die Grafikdarstellungen in dem Ordner <i>XSDChart</i> .
<i>XSD4JServlet2</i>	Wenn der Parser das Schema <i>BootstrappedSchema11</i> wählt, ruft dieses Servlet das XSD-Dokument auf und überprüft alle XSD-Dokumente zusammen. Er kann die Testfälle von den Ordnern <i>tests</i> , <i>test1_w3c</i> , <i>test2_11</i> , <i>test3_xerces_j</i> , und <i>test4_saxon</i> überprüfen. Außerdem speichert es die Grafikdarstellungen in dem Ordner <i>XSDChart</i> .
<i>XSD4JTestServlet</i>	Wenn der Parser das Schema <i>BootstrappedSchema11</i> wählt, ruft dieses Servlet das XSD-Dokument auf und überprüft einzelne XSD-Dokumente separat. Er kann die Testfälle von dem Ordner <i>test2_11</i> überprüfen. Dieser Ordner <i>test2_11</i> enthält alle Testfälle, die von der XSD4J Bibliothek unterstützt werden.
<i>XSD4JTestsServlet</i>	Wenn der Parser das Schema <i>BootstrappedSchema10</i> wählt, ruft dieses Servlet das XSD-Dokument auf und überprüft einzelne XSD-Dokumente separat. Er kann die Testfälle von dem Ordner <i>tests</i> überprüfen. Dieser Ordner <i>tests</i> enthält alle Testfälle, die von der XSD4J Bibliothek unterstützt werden.
<i>XSD4Jdownload-Servlet</i>	Dieses Servlet unterstützt <i>http protocol</i> , er kann das XSD-Dokument von der Webseite in den lokalen Ordner herunterladen. Dann speichert es dieses XSD-Dokument in dem Ordner <i>temp</i> und führt die Aufgaben weiter aus. Dieser Ordner <i>temp</i> bleibt immer erhalten, ohne dass der Server neu gestartet wird. Die Funktion von <i>delettemp.jsp</i> kann beliebig diesen Ordner <i>temp</i> löschen.

Tabelle 6.9: Information der Servlets

Tabelle 6.10 zeigt die Information der JSPs im Test-Werkzeug XSD4JTest.

JSPs	Beschreibung
<i>abgeben.jsp</i>	Diese JSP-Seite ruft das ausgewählte XSD-Dokument auf, dann repräsentiert der Browser wieder dieses XSD-Dokument durch Applet <i>ResourceTest</i> . Schließlich ruft sie den Servlet <i>XSD4JServlet2</i> auf. Das ist ein einzelner Testfall von XSD 1.1.
<i>abgebens.jsp</i>	Diese JSP-Seite ruft das ausgewählte XSD-Dokument auf, dann repräsentiert der Browser wieder dieses XSD-Dokument durch Applet <i>ResourcesTest</i> . Schließlich ruft er das Servlet <i>XSD4JServlet1</i> auf. Das ist ein einzelner Testfall von XSD 1.0.
<i>about.jsp</i>	Diese JSP-Seite informiert über diese Diplomarbeit.
<i>deletetemp.jsp</i>	Diese JSP-Seite kann den Ordner <i>temp</i> löschen, in dem die XSD-Dokumente aus der Webseite gespeichert werden.
<i>home.jsp</i>	Diese JSP-Seite notiert die Implementierung dieser Arbeit.
<i>index.jsp</i>	Diese JSP-Seite ist die hauptsächliche Seite über das XSD4JTest Testwerkzeug, diese JSP-Seite enthält einzelne Testfälle von XSD1.0 und XSD 1.1, auch die mehreren Testfälle der verschiedenen Testfälle aus W3C, Xercers-J und Saxon.
<i>note.jsp</i>	Notiert die Bugs der Implementierung.
<i>xsd.jsp</i>	Diese JSP-Seite wird von <i>index.jsp</i> aufgerufen.

Tabelle 6.10: Information der JSPs

Tabelle 6.11 zeigt die Information des Ordners im Test-Werkzeug XSD4JTest.

Testfälle	Beschreibung
<i>tests</i>	Dieser Ordner enthält die Testfälle von XSD 1.0.
<i>test1_w3c</i>	Dieser Ordner enthält die Testfälle von W3C. Diese Testfälle bestehen aus den Datentypen und den Strukturen von XSD 1.0 und XSD 1.1.
<i>test2_11</i>	Dieser Ordner enthält die Testfälle von XSD 1.1, die in der Implementierung der XSD4J Bibliothek überprüft werden.
<i>test3_xercers_j</i>	Dieser Ordner enthält die Testfälle von Xercers-J, die in der neuesten Version Xercers-J 2_10_0 (SVN Version 926149) heruntergeladen werden.
<i>test4_saxon</i>	Dieser Ordner enthält die Testfälle von Saxon EE, die in der neuesten Version Saxon EE 2.9.2.0.2 heruntergeladen werden.
<i>test10</i>	Enthält alle Testfälle von XSD 1.0.
<i>test11</i>	Enthält alle Testfälle von XSD 1.1.

Tabelle 6.11: Information des Ordners

## 6.6 Zusammenfassung

In diesem Kapitel wurde zunächst das Konzept der Testfälle der XSD4J Bibliothek dargestellt. Es gibt zwei Methoden: den einzelnen Testfall und mehrere Testfälle. Ein einzelner Testfall bedeutet, dass der Testfall separat in der XSD4J Bibliothek überprüft wird und mehrere Testfälle bedeuten, dass die Testfälle zusammen in der XSD4J Bibliothek überprüft werden.

Dann wurde die Implementierung des Test-Werkzeugs XSD4JTest nach dem MVC-Entwurfs-Muster erläutert. Das Test-Werkzeug XSD4JTest kann einerseits automatisch einzeln überprüfbare Testfälle und mehrere überprüfbaren Testfälle der XSD-Dokumente unterstützen und andererseits kann es die Testfälle der XSD-Dokumente aus der Webseite direkt mit dem URL überprüfen.

Des Weiteren wurden die Testfälle in der XSD4J Bibliothek überprüft. Diese Testfälle stammten aus der vervollständigten Implementierung für die XSD4J Bibliothek in dieser Arbeit, W3C, Xerces-J und Saxon EE. Durch die Prüfung dieser Testfälle können wir sehen, dass die XSD4J Bibliothek einzelne Testfälle schneller als mehrere Testfälle für das gleiche XSD-Dokument bearbeiten konnte. Außerdem ist die Laufzeit der Testfälle abhängig von der Komplexität des überprüften XSD-Dokuments. Je höher der Komplexität des XSD-Dokuments ist, desto langsamer läuft es, wenn die Datengröße identisch ist.

Daher kann festgehalten werden: Die XSD4J Bibliothek bearbeitet das XSD-Dokument, ihre Leistungen sind abhängig von der Bearbeitungsmethode und der Dokumentstruktur.

Schließlich wurden die APIs, Servlets und JSPs von XSD4JTest beschrieben.

Um eine höhere Leistung zu erreichen, wird die XSD4J Bibliothek noch erweitert. Im nächsten Kapitel wird die Erweiterung der XSD4J Bibliothek bearbeitet.

# Kapitel 7

## Erweiterung der XSD4J Bibliothek für SAWSDL

In diesem Kapitel wird die XSD4J Bibliothek für semantische Annotationen von Schemakonstrukten erweitert, implementiert und mit den Testfällen überprüft, um die Beschreibung der Datentypen von SAWSDL im XML-Schema zu unterstützen.

### 7.1 Überblick von SAWSDL

Die Schnittstellenbeschreibungssprache Semantic Annotations for WSDL and XML Schema (SAWSDL) [40] ist eine Weiterentwicklung von WSDL-S und stellt eine Reihe von Erweiterungen für die Web Service Description Language (WSDL) zur Verfügung, mit denen sich Web Service-Schnittstellen mit formaler Semantik aus externen Ontologien anreichern lassen. Dadurch soll ein automatisiertes Auffinden, Abgleichen, Zusammenfügen und Aufrufen von Web Services möglich werden [41].

Es gibt zwei wesentliche Mechanismen von Annotation in der Abbildung 7.1:

- Ein erweitertes Attribut ist *modelReference*, um die Assoziation zwischen einer WSDL oder XML-Schema Komponente und einem Konzept in einigen semantischen Modellen zu präzisieren. Es wird verwendet, um XML-Schema-Definitionen, Element-Deklarationen und Attribut-Deklarationen ebenso wie WSDL-Schnittstellen, Operationen und Fehler zu kommentieren.

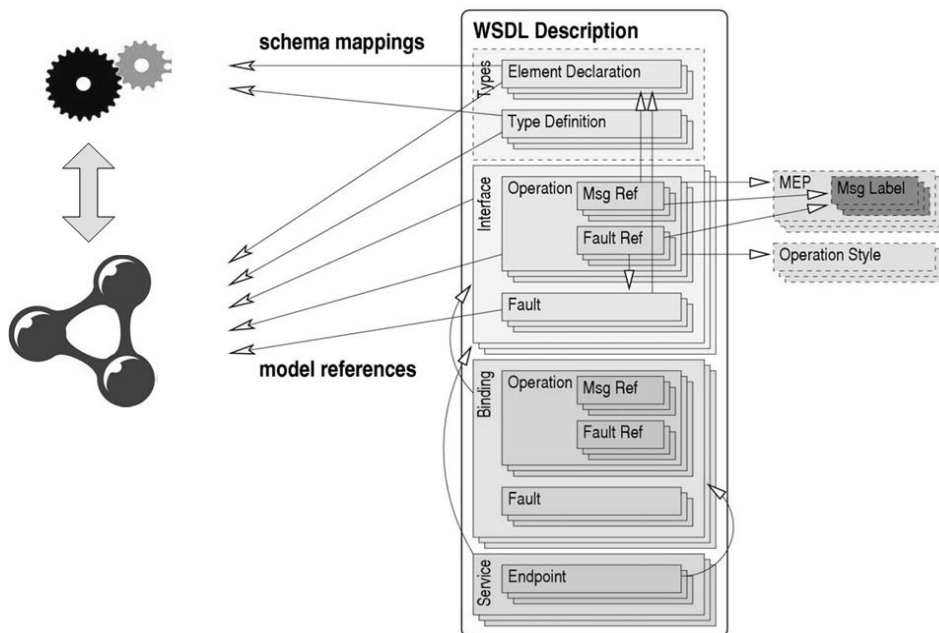


Abbildung 7.1: SAWSDL im Bild [42]

- Zwei Attribute der Erweiterung heißen *liftingSchemaMapping* und *loweringSchemaMapping*, die die Elementdeklarationen des XML-Schemas und Typdefinitionen zur zugeordneten Spezifikation (Mappings) zwischen semantischen Daten und XML hinzufügen. Die Daten im Semantic Web werden durch semantische Modelle vertreten. Die Abbildung 7.2 beschreibt den grafisch diesen Datenaustausch.

Es gibt zwei wesentliche Modelle im W3C Standard: Resource Description Framework (RDF) und Web Ontology Language (OWL) [45]. Die RDF wird verwendet, um Informationen darzustellen und das Wissen auszutauschen. OWL wird verwendet, um Ontologien zu definieren. Das Ziel ist es, Websuchen und Wissensmanagement zu unterstützen [55].

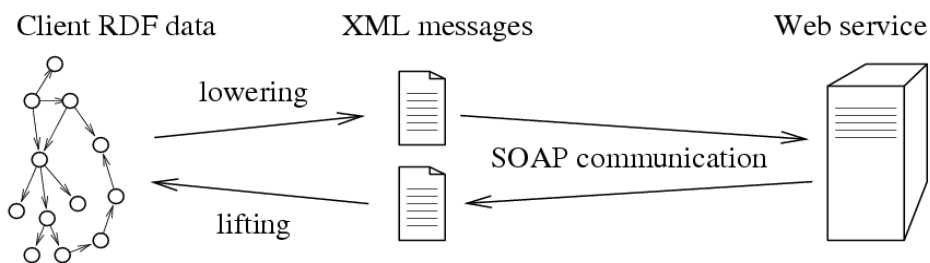


Abbildung 7.2: Lifting und Lowering [42]

Das semantische Model ist nicht innerhalb von WSDL definiert. SAWSDL ist durch die Attributdefinitionen in WSDL und XSD zu realisieren, um Web Service und Semantic Web zu verbinden.

Quelle: <http://www.w3.org/TR/sawSDL/#Using>

## 7.2 Die Liste der Merkmale von SAWSDL im XSD

Um die Funktion von SAWSDL zu unterstützen, muss die XSD4J Bibliothek um folgenden Elemente oder Attribute erweitert werden.

Tabelle 7.1 listet die verfügbaren Attribute in SAWSDL auf. Diese Attribute werden mit den entsprechenden Elementen und Attributen in XSD erweitert [18].

Name	Elemente oder Attribute
sawSDL:modelReference	<simpleType>, <complexType>, <element>, <attribute>
sawSDL:liftingSchemaMapping	<simpleType>, <element>, <complexType>
sawSDL:loweringSchemaMapping	<simpleType>, <element>, <complexType>

Tabelle 7.1: Die Erweiterung der Elemente oder Attribute von XSD4J

## 7.3 XSD4J-Erweiterung für SAWSDL

Die XSD4J Bibliothek SVN-Version (Standard: 1426) hat vor dieser Bearbeitung meist die existierten Charaktere wie einfache Typen, komplexe Typen, Einschränkung eines Typs, Elementdefinition und Attributedefinition in XSD 1.0 unterstützt. Nach dieser Bearbeitung ist sie anhand der XSD 1.1 Spezifikation ( Draft 3 Dec. 2009) entwickelt geworden. Die Ergänzungen und neuen Funktionen wurden im Kapitel 5 vorgestellt. Das Test-Werkzeug XSD4JTest wurde im Kapitel 6 eingeführt. Der folgende Abschnitt wird die Erweiterung der XSD4J Bibliothek entwickeln. Die Hauptaufgabe bezieht sich hier auf SAWSDL. Die XSD4J-Erweiterung von SAWSDL enthält die Unterstützung von den Attributen *modelReference*, *liftingSchemaMapping* und *loweringSchemaMapping*. Namensraum ist *xmlns:sawSDL="http://www.w3.org/ns/sawSDL"* für SAWSDL.

### 7.3.1 modelReference.

*modelReference* kann die Elemente `<xsd:simpleType>`, `<xsd:complexType>`, `<xsd:element>` und `<xsd:attribute>` als Attribut verwenden. Deshalb muss das Attribut *modelReference* zu diesen Elementen in der XSD4J Bibliothek ergänzt werden. Listing 7.3.1 zeigt, dass das Attribut *modelReference* mit dem Präfix *sawSDL* als ein Attribut des einfachen Datentyps deklariert wird. Der einfache Datentyp *confirmation* wird als ein URI von *sawSDL:modelReference* eingeführt.

Listing 7.3.1: Erweiterung für das Attribut modelReference

```

...
<xsd:simpleType name="confirmation"
sawSDL:modelReference=
"http://www.w3.org/2002/ws/sawSDL/spec/ontology/purchaseorder#OrderConfirmation">
  <xsd:restriction base="xsd:string">
    ...
  </xsd:restriction>
</xsd:simpleType>
...

```

Quelle: <http://www.w3.org/TR/sawSDL/#modelReference>

### 7.3.2 loweringSchemaMapping und liftingSchemaMapping

*loweringSchemaMapping* und *liftingSchemaMapping* können von den Elementen `<xsd:simpleType>`, `<xsd:complexType>` und `<xsd:element>` auch als Attribut verwendet werden. Deshalb müssen die Attribute *loweringSchemaMapping* und *liftingSchemaMapping* in der XSD4J Bibliothek ergänzt werden. Listing 7.3.2 zeigt, dass das Attribut *loweringSchemaMapping* mit dem Präfix *sawSDL* als ein Attribut des komplexen Datentyps deklariert wird. Als komplexer Datentyp *itemType* wird ein URI von *sawSDL:loweringSchemaMapping* eingeführt. Dieses URI ist ein XSLT-Dokument, das nach der Abbildung des XSD-Dokuments realisiert wird.

Listing 7.3.2: Erweiterung für das Attribut `loweringSchemaMapping`

```
...
<xsd:complexType name="itemType"
  sawsdl:liftingSchemaMapping="http://example.org/mapping/ItemType2Ont.xslt">
  <xsd:sequence>
    ...
  </xsd:sequence>
  <xsd:attribute name="ItemID" type="xsd:string"/>
</xsd:complexType>
...
```

Quelle: <http://www.w3.org/TR/sawSDL/#annotateXSDSchemaMapping>

### 7.4 Zusammenfassung

In diesem Kapitel wurde zunächst die Einführung von SAWSDL dargestellt. SAWSDL konnte durch die Erweiterung von XSD bearbeitet werden, um den Datenaustausch zwischen Web Service und Semantic Web zu erreichen. Diesen Datenaustausch gibt es in zwei Arten: *modereference* oder *lifterSchemaMapping* und *loweringSchemaMapping*. Das Attribut *modereference* konnte die Assoziation zwischen einer WSDL oder XML-Schema Komponente und einem Konzept in dem eigenen semantischen Modell präzisieren. Die Attribute *lifterSchemaMapping* und *loweringSchemaMapping* konnten die Elementdeklarationen des XML-Schemas und Typdefinitionen zur zugeordneten Spezifikation (Mappings) zwischen semantischen Daten und XML hinzufügen. Nach der Erweiterung von diesen beiden Arten wurde von der XSD4J Bibliothek das XSD-Dokument bearbeitet, das die Syntax von SAWSDL enthält. Das bereichert die Funktion der XSD4J Bibliothek.



# Kapitel 8

## Zusammenfassung und Ausblick

Dieses Kapitel fasst in 8.1 noch einmal den Inhalt dieser Arbeit zusammen, diese bezieht sich auf die Vervollständigung, Testfälle, und Erweiterung der XSD4J Bibliothek mit XSD 1.1. Abschnitt 8.2 gibt einen Ausblick auf die weitere Entwicklung der XSD4J Bibliothek und dem Test-Werkzeug XSD4JTest.

### 8.1 Zusammenfassung

Dieser Abschnitt führt die Ergebnisse dieser Arbeit zusammen und enthält jeweils Verweise auf die Abschnitte, die sich mit ihnen befassen. Die Ergebnisse beziehen sich auf die drei Aufgaben der Arbeit: Vervollständigung, Testfälle und Erweiterung der XSD4J Bibliothek.

#### Vervollständigung der XSD4J Bibliothek

Nach den theoretischen Grundlagen (Kapitel 2), in denen die Relevanz des Themas für das XML-Schema beschrieben wird, geht Kapitel 3 auf das Inhaltsmodell mit XML-Schema ein. In diesem Kapitel geht es um die Struktur und den Datentyp von XSD. Das XSD-Dokument wird anhand der Definitionen dieser Strukturen und Datentypen aufgebaut. Die Strukturdefinition enthält Element- und Attributdeklaration, globale und lokale Deklarationen, Kompositoren, Partikel, mehrere Namensräume in XSD, Identitätseinschränkungen, Definition von Element- und Attributgruppen, einfache Typdefinition und komplexe Typdefinition sowie Wildcard. Die Datentypdefinition bezieht sich auf vordefinierte einfache Typen: primitive Datentypen und abgeleitete einfache Datentypen, komplexe Datentypen und ihre Verwendungen beim XSD-Dokument. Auf der anderen Seite wurden die atomaren Datentypen, die Listen- und Vereinigungsdatentypen und die Facetten erläutert. Die Facetten spielen eine Rolle für die Einschränkung der Datentypen im XSD-Dokument. Das Verstehen der relevanten Kenntnisse über die XSD-Spezifikation ist eine Voraussetzung, um die XSD4J Bibliothek zu vervollständigen. Die W3C Working Group war verantwortlich für die Entwicklung einer Spezifikation des standardisierten XML-Schemas. Am Ende dieses Kapitels wurden die anderen Schemasprachen RELAX NG und Schematron kurz eingeführt und beide mit dem XML-Schema verglichen. Im Kapitel 4 folgt die Analyse der Architektur und die Umsetzung der XSD4J Bibliothek, in denen unter anderem auf DOM, XSD4J mit DOM, die innere Architektur von XSD4J sowie XSDParser, XSDDumper sowie den Implementierungsablauf von Parser und Dumper in den Klassen XSDParser und XSDDumper in der XSD4J Bibliothek anhand eines Beispiels eingegangen wird. Besonders wurden die Klassen der XSD4J Bibliothek, die APIs der Klassen XSDParser und XSDDumper sowie die Umsetzung zwischen den Attributen und Elementen in XSD 1.1 und Java-Objekte tabelliert. Am Ende dieses Kapitels wurde die Beschreibung der vervollständigten Elemente und Attribute sowie der neuen Datentypen von XSD 1.1

zusammengefasst. Die Arbeit mit XSD 1.1 in der XSD4J Bibliothek wurde in Kapitel 5 anhand der neuen Merkmale aus der XSD 1.1 Spezifikation herausgearbeitet. Diese Merkmale enthalten die Strukturen, in denen unter anderem auf die Steuerung des Einschusses, Assertions, Condition Type Assigement (CTA), Open-Content, Schema-wide-Attribut und Element `<xsd:all>` beschrieben wird, die Datentypen, die sich auf `<xsd:anyAtomicType>`, `<xsd:precisionDecimal>`, `<xsd:dateTimeStamp>`, `<xsd:yearMonthDuration>` und `<xsd:dayTimeDuration>` beziehen, und die Facetten, die `<xsd:assertion>`, `<xsd:explicitTimezone>`, `<xsd:minScale>` und `<xsd:maxScale>` behandeln. Darüber hinaus werden die Anwendungen der verfügbaren neuen Strukturen und Facetten in den Elementen von XSD 1.1 tabelliert. In dieser Tabelle kann man sofort die Aktualisierung von XSD 1.1 finden. Der Überblick nach der vervollständigten Implementierung der XSD4J Bibliothek wird im Anschluss dieses Kapitel tabelliert.

### Testfälle in der Test-Werkzeug XSD4JTest

Im Kapitel 6 beschreibt die mittels dem Test-Werkzeug XSD4JTest überprüften Testfällen. Diese wurden bei der Untersuchung in der XSD4J Bibliothek in zwei Gruppen getrennt: Einzelfall, der separat überprüft wird, und mehrere Testfälle, die zusammen überprüft werden. Das Test-Werkzeug XSD4JTest überprüft automatisch einzelne und mehrere Testfälle für XSDParser und XSDDumper in der XSD4J Bibliothek. Diese Testfälle bestanden aus der vervollständigten Implementierung für die XSD4J Bibliothek in dieser Arbeit, W3C, Xerces-J und Saxon EE. Als Ergebnis der beiden Untersuchungen wird angegeben, dass die Bearbeitung des XSD-Dokuments in der XSD4J Bibliothek abhängig von der Bearbeitungsmethode, die sich auf Einzelfall und mehrere Testfälle bezog, und der Dokumentstruktur, die die Komplexität des XSD-Dokuments beschrieb, war. Ein Einzelfall des gleichen XSD-Dokuments ist schneller als mehrere Testfälle und je höher die Komplexität des überprüften XSD-Dokuments ist, desto langsamer läuft es, wenn die Datengröße identisch ist.

Darüber hinaus werden am Ende dieses Kapitel die APIs, Servlets und JSPs von XSD4JTest kurz beschrieben.

### XSD4J-Erweiterung für SAWSDL

Um die verschiedenen Funktionen zu erreichen, wird die XSD4J Bibliothek noch erweitert. Im Kapitel 7 wird die Erweiterung der XSD4J Bibliothek für SAWSDL beschrieben. SAWSDL ist eine Spezifikation, die die Schnittstelle des Datenaustausches zwischen Web Service und Semantic Web definiert. Dadurch soll ein automatisiertes Auffinden, Abgleichen, Zusammenfügen und Aufrufen von Web Services mit formaler Semantik aus externen Ontologien erreicht werden. Durch die Erweiterung der Elemente und Attribute, in denen das XML-Schema des WSDL-Dokuments in Web Service definiert wird, kann diese Interaktion erreicht werden. Diese Erweiterungen existieren in zwei Mechanismen: die Erweiterung des Attributs *modelReference* und die Erweiterung der Attribute *liftingSchemaMapping* und *loweringSchemaMapping*.

Nach der Erweiterung für SAWSDL konnte das XSD-Dokument mit der SAWSDL-Syntax von der XSD4J Bibliothek bearbeiten werden. Das bereichert die Funktion der XSD4J

Bibliothek.

## 8.2 Ausblick

Zum Schluß dieser Arbeit sollte noch ein Ausblick der XSD4J Bibliothek und dem Test-Werkzeug XSD4JTest gegeben werden:

Über der XSD4J Bibliothek:

- XSD 1.1 verfügt mit der XPath-Expression [49] über eine stärkere Leistung, deshalb kann die Unterstützung von XPath-Expression im XSD 1.1 weiter entwickelt werden.
- Bisher kann das Element `<xsd:annotation>` nach der aktuellen Spezifikation XSD 1.1 mehrfach von dem Wurzelement `<xsd:schema>` referenziert werden. Andere Elemente können nur einmal bei ihm referenziert werden. Um einen Konflikt zwischen Java-Objekten in der XSD4J Bibliothek zu vermeiden und einen effizienten Parser zu erhalten, sollte erforscht werden, ob es effektivere Umsetzungsmöglichkeiten gibt.
- Die XSD4J Bibliothek kann zurzeit nur XSD-Dokument bearbeiten. Daher steht die Frage nach der Umsetzung in einem anderen Schema-Dokument, z.B. RELAX-NG-Schema und Schematron, weiterhin im Raum.
- Die Erweiterung der XSD4J Bibliothek ist ebenso wichtig, nicht nur bzgl. SAWSDL, sondern auch hinsichtlich anderer Möglichkeiten.
- Die Entwicklung wurde unter Java Version „1.6.0\_16“ implementiert. Ob die XSD4J Bibliothek effektiv unter einer anderen Java Version laufen kann, muss noch untersucht werden.
- Das API-Dokument kann weiter verbessert werden.

Über das Test-Werkzeug XSD4JTest:

- XSD4JTest ist nur einfaches Test-Werkzeug, das die Testfälle von Parser und Dumper für die XSD4J Bibliothek bedient. Es ist abhängig von JfreeChart 1.0.13, um die Statistik zu repräsentieren. XSD4JTest konnte für andere Funktionen in der XSD4J Bibliothek als paralleles Test-Werkzeug entwickelt werden, z.B. zur Validierung des XML-Dokuments.



# Literaturverzeichnis

- [1] Tim, Bray; Jean Paoli; C. M. Sperberg-McQueen; Eve Maler; François, Yergeau; John, Cowan: *Extensible Markup Language (XML) 1.1 (Second Edition)*, W3C Recommendation , 16. Aug. 2006.
- [2] Vonhoegen, Helmut: *Einstieg in XML: Grundlagen, Praxis, Referenzen ; für Entwickler und XML-Einsteiger;Formatierung, Transformation, Schnittstellen ; XML Schema, DTD, XSLT 1.0 2.0, XPath 1.0 2.0, DOM, SAX,SOAP, Open XML*, 2007, Galileo Press Verlag, Bonn, 4. aktualisierte Aufl., ISBN 978-3-8362-1074-4.
- [3] Frank ,P. Coyle: *XML, Web Services and the Changing Face of Distributed Computing*; April.2002, ACM New York, NY, USA.
- [4] Rick, Jelliffe: *Family Tree of Schema Languages (v6)*, Nov 2006.  
<[http://www.oreillynet.com/xml/blog/2006/11/family\\_tree\\_of\\_schema\\_language\\_1.html](http://www.oreillynet.com/xml/blog/2006/11/family_tree_of_schema_language_1.html)>
- [5] Charles ,Goldfarb und Paul ,Prescod: *Das XML-Handbuch : Anwendungen, Produkte, Technologien*, 2000, Addison-Wesley Verlag, Münch, 2.aubl.
- [6] Guido, Krüger; Stark, Thomas: *Handbuch der Java-Programmierung*, 2009, Addison-Wesley Verlag, 5. Aufl., München.
- [7] Tim Bray, Textuality; Dave Hollander; Andrew Layman; Richard Tobin; Henry S. Thompson: *Namespaces in XML 1.0 (Third Edition)*, W3C Recommendation, 8. December 2009.
- [8] James, Clark; Steve, DeRose : *XML Path Language (XPath) Version 1.0*, W3C Recommendation 16. November 1999.  
<<http://www.w3.org/TR/xpath/>>
- [9] Erik, Wilde: *XML Path Language (XPath)*, 23.Sep.2008, UC Berkeley School of Information.
- [10] Eckstein, Rainer; Eckstein, Silke: *XML und Datenmodellierung: XML Schema und RDF zur Modellierung von Daten und Metadaten einsetzen*, 2004, dpunkt.verlag Verlag, Heidelberg, 1. Aufl., ISBN 3-89864-222-4.
- [11] Thompson, S.; Beech, David; Maloney, Murray; Mendelsohn, Noah: *XML Schema Part 1: Structures Second Edition*,W3C Recommendation, 2004.  
<<http://www.w3.org/TR/2004/REC-xmlschema-1-20041028/>>

- [12] Stefanie, Dipper: *XML Fortgeschrittenes: DTDs, XML Schema, XML in der CL*, 3. Nov. 2004.
- [13] Mario, Jeckle: *Vorlesung in XML*, 2004.  
<http://www.jeckle.de/vorlesung/xml/script.html>
- [14] Hall, Marty : *Core servlets und JavaServer Pages*, 2001, Markt +Technik Verlag, München, ISBN 3-8272-5945-2.
- [15] Mukul Gandhi: *XSD 1.1: xs:precisionDecimal, assertions and Xerces-J updates*.  
17. Apri. 2010.  
<<http://mukulgandhi.blogspot.com/>>
- [16] Roger, L. Costello: <http://www.xfront.com>, 14 Oct, 2009.
- [17] Fallside, David C.; Walmsley, Priscilla: *XML Schema Part 0: Primer Second Edition*, W3C Recommendation, 2004.  
<<http://www.w3.org/TR/xmlschema-0/>>;  
<<http://www.w3.org/TR/2004/REC-xmlschema-0-20041028/>>
- [18] Cristina, Baroglio; Piero A. Bonatti; Jan, Maluszynski; Massimo, Marchiori; Axel, Polleres; Sebastian, Schaffert: *Reasoning Web: 4th International Summer School 2008*, Venice Italy, September 7-11, 2008, Tutorial Lectures, Verlag. Springer, 1 Auflag.
- [19] Biron, Paul V.; Malhotra, Ashok : *XML Schema Part 2: Datatypes Second Edition*, W3C Recommendation, 2004.  
<<http://www.w3.org/TR/2004/REC-xmlschema-2-20041028/>>
- [20] Wikipedia: *Schema (Informatik)*, 2009.  
<[http://de.wikipedia.org/wiki/Schema\\_%28Informatik%29](http://de.wikipedia.org/wiki/Schema_%28Informatik%29)>
- [21] Eric, van der Vlist: *XML Schema*, Januar 2003, Verlag: O'Reilly; Auflage: 1, ISBN-13: 978-3897213456.
- [22] Vonhoegen, Helmut: *Einstieg in XML : Grundlagen, Praxis, Referenzen ; für Entwickler und XML-Einsteiger; Formatierung, Transformation, Schnittstellen ; XML Schema, DTD, XSLT 1.0 2.0, XPath 1.0 2.0, DOM, SAX, SOAP, Open XML*, 2009, Galileo Press Verlag, Bonn, 5. aktualisierte Aufl., ISBN 978-3-8362-1367-7.
- [23] Rahul, Srivastava: *XML Schema: Understanding Structures*, Oracle Technology Network, 2009.  
<[http://www.oracle.com/technology/pub/articles/srivastava\\_structures.html](http://www.oracle.com/technology/pub/articles/srivastava_structures.html)>

- [24] Rahul, Srivastava: *XML Schema: Understanding Datatypes*, Oracle Technology Network, 2009.  
<[http://www.oracle.com/technology/pub/articles/srivastava\\_datatypes.html](http://www.oracle.com/technology/pub/articles/srivastava_datatypes.html)>
- [25] James, Clark: *RELAX NG and W3C XML Schema*, 4 Jun. 2002.  
<<http://www.imc.org/ietf-xml-use/mail-archive/msg00217.html>>
- [26] Jochen, Schwarze: *RELAX NG als XML-Schemasprache*,  
Der Beauftragte der Bundesregierung für Informationstechnik.  
<[http://www.cio.bund.de/kbst\\_forum/showthread.php?t=154](http://www.cio.bund.de/kbst_forum/showthread.php?t=154)>,  
cit GmbH, <<http://www.cit.de>>
- [27] P. Warmusley: *Definitive XML Schema*, Prentice Hall, 2002.
- [28] Wiki: *Schematron* 2010.  
<<http://de.wikipedia.org/wiki/Schematron>>;  
<<http://schematron.com/overview.html>>
- [29] Dare, Obasanjo: *Verbessern der Gültigkeitsprüfung von XML-Dokumenten mit Schematron*, 10. Dez 2004.  
<<http://msdn.microsoft.com/de-de/library/cc431159.aspx>>
- [30] Josef, Spillner: *Project Dynvocation*, 2008.  
<<http://dynvocation.selfip.net/>>
- [31] Josef, Spillner: *Diplomarbeit zum Thema Entwicklung eines Editors zum Entwurf von Benutzerschnittstellen für Web Services auf Basis der abstrakten UI-Beschreibungssprache WSGUI*, 14. September 2006.
- [32] Tim, Bray; Jean, Paoli; C. M. Sperberg-McQueen; Eve, Maler; François, Yergeau und John, Cowan: *Extensible Markup Language (XML) 1.1 (Second Edition)*, W3C Recommendation, Aug.2006.  
<<http://www.w3.org/TR/xml11/#sec-well-formed>>
- [33] Dirk, Louis ; Peter, Müller: *Java 6: Praxis der objektorientierten Programmierung* , 2007, Markt & Technik in Pearson Education Deutschland Verlag, München.
- [34] Heinz, Wittenbrink: *XML: Wissen, das sich auszahlt*, 2003,  
TEIA Lehrbuch Verlag, Berlin, ISBN 3-935539-69-X.
- [35] Josef, Spillner: *Die Architektur von XSD4J*, 2009.  
<<http://dynvocation.selfip.net/docs/xsd4j-arch.png>>

- [36] C. M. Sperberg-McQueen: *XML Schema (XSD) 1.1*, black Mesa Technologies , 27 July. 2009.  
<<http://www.blackmesatech.com/2009/07/xsd11>>
- [37] David, Peterson; Shudi (Sandy), Gao; 高殊镝; Ashok, Malhotra; C. M. Sperberg-McQueen; Henry S. Thompson: *W3C XML Schema Definition Language (XSD) 1.1 Part 2: Datatypes*; Dec.2009.  
<<http://www.w3.org/TR/2009/WD-xmlschema11-2-20091203/type-hierarchy-200901.longdesc.html>>
- [38] Hall, Marty : *Core servlets und JavaServer Pages*, 2001, Markt +Technik Verlag, München, ISBN 3-8272-5945-2.
- [39] Basham, Bryan; Sierra, Kathy; Bates, Bert: *Servlets und JSPs von Kopf bis Fuß*, 2009, O'Reilly Verlag, Beijing ; Köln, 2. Aufl., ISBN ISBN 978-3-89721-873-4.
- [40] Joel, Farrell; Holger, Lausen: *Semantic Annotations for WSDL and XML Schema*, W3C Recommendation, 28. Aug,2007.
- [41] Jacek, Kopecký: *SAWSDL Status and relation to WSMO*, 14.March.2007, DERI Innsbruck University of Innsbruck.
- [42] Jacek, Kopecký: *Semantic Annotations for WSDL and XML Schema*, WWW2007, W3C Track, Banff, May. 2007.  
<<http://www.w3.org/2007/Talks/www2007-sawSDL/2007-05-sawSDL.html>>
- [43] ZHANG,Yu;LI,Fa: *Partial Validation of XML Document Based on XML Schema with Xerces-J*, Department of Computer Science, University of Science&Technology of China, Hefei, China, Aug.2005.
- [44] Delima, Neil; Gao, Sandy; Glavashevich, Michael; Noaman, Khaled: *XML Schema 1.1, Part 2: An introduction to XML Schema 1.1*, 2009.  
<<http://www.ibm.com/developerworks/xml/library/x-xml11pt2/>>
- [45] Iris, Braun : *SOA – Entwicklung verteilter Systeme auf Basis serviceorientierter Architekturen*; Kapitel 3. Implementierung von Web Services und Clients, 2008.
- [46] Mario, Jeckle: *XML-Schemasprachen und W3Cs XML-Schema*, DaimlerChrysler Forschungszentrum, Ulm, 2002.
- [47] Torsten, Horn: *Ingenieurbüro für Softwareentwicklung und Javatechnologien-XSD(XML Schema Definition)*, 2009.



- [48] Costello, Roger L.: *XML Schema 1.1 Tutorial*, 2009.
- [49] Costello, Roger L.: *XPath Tutorial*, 2009.
- [50] Vonhoegen, Helmut: *Einstieg in XML : Grundlagen, Praxis, Referenzen ; für Entwickler und XML-Einsteiger;Formatierung, Transformation, Schnittstellen ; XML Schema, DTD, XSLT 1.0 2.0, XPath 1.0 2.0, DOM, SAX, SOAP, Open XML*, 2009, Galileo Press Verlag, Bonn, 5. aktualisierte Aufl., ISBN 978-3-8362-1367-7.
- [51] Microsoft: *XML Schemas (XSD) Reference*, ,msdn,2009.  
<[http://msdn.microsoft.com/en-us/library/ms256235\(v=VS.100\).aspx](http://msdn.microsoft.com/en-us/library/ms256235(v=VS.100).aspx)>
- [52] Anderson, Richard; Birbeck, Mark; Kay, Michael; Livingstone, Steven; Loesgen, Brian; Martin, Didier; Mohr,Stephen; Ozu, Nikola; Bruce, Peat; Pinnock, Jonathan; Stark, Peter; Williams, Kevin: *Professional XML*, 2000, Wrox Press Ltd. Verlag, Birmingham, ISBN ISBN 1-861003-11-0.
- [53] Oliver, Vogel; Ingo, Arnold; Arif, Chughtai; Edmund, Ihler; Timo, Kehrler; Uwe, Mehlig und Uwe, Zdun: *Software-Architektur: Grundlagen - Konzepte - Praxis*, Spektrum Akademischer Verlag, Heidelberg, 2 Aufl. 2009, ISBN 978-3-8274-1933-0.
- [54] *The Apache Xerces Project*, 2007.  
<<http://xerces.apache.org/#xerces2-j>>
- [55] Mark, D. Hansen: *SOA Using Java Web Services* Prentice Hall PTR , 2007.
- [56] SAXON The XSLT and XQuery Processor: *SAXON EE 9.2.*, 2010.  
<<http://saxon.sourceforge.net/>>
- [57] TU Dresden Fakultät Informatik: *Projekt Dynvocation - Regular Expressions Instances and XML Schema*, 2009.  
<<http://dynvocation.selfip.net/regexinstant/>>
- [58] Kazakos, Wassilios; Schmidt,Andreas; Tomczyk, Peter: *Datenbanken und XML-Konzepte-Anwendungen, Systeme*, Verlag Springer, 2002, ISBN ISBN 3-540-41956-X.
- [59] Erik, Wilde: *XML Namespaces*, 18.Sep.2008, UC Berkeley School of Information.
- [60] Biron, Paul V.; Malhotra, Ashok: *XML Schema Part 2: Datatypes Second Edition,W3C Recommendation*, 2004.

- <<http://www.w3.org/TR/2004/REC-xmlschema-2-20041028/>>
- [61] Delima, Neil; Gao, Sandy; Glavassevich, Michael; Noaman, Khaled: *XML Schema 1.1, Part 1: An introduction to XML Schema 1.1*, 2008.  
<<https://www.ibm.com/developerworks/library/x-xml11pt1>>
- [62] Delima, Neil; Gao, Sandy; Glavassevich, Michael; Noaman, Khaled: *XML Schema 1.1, Part 3: An introduction to XML Schema 1.1*, 2009.  
<<http://www.ibm.com/developerworks/xml/library/x-xml11pt3/>>
- [63] Fallside, David C.; Walmsley, Priscilla: *XML Schema Part 0: Primer Second Edition*, W3C Recommendation, 2004.  
<<http://www.w3.org/TR/2004/REC-xmlschema-0-20041028/>>
- [64] Object Management Group, Jul 2009.  
<[http://www.omg.org/gettingstarted/omg\\_idl.htm](http://www.omg.org/gettingstarted/omg_idl.htm)>
- [65] Elliott Rusty, Harold: *Die XML-Bibel*, MITP-Verl. Auflage, 1. Aufl., Bonn, 2000, ISBN 3-8266-0627-2.
- [66] Felix, Michel: *Representation of XML Schema Components*, Master Thesis, mai 2007, School of Information, University of California, Berkeley.
- [67] Pott, Oliver; Wielage, Gunter: *xml-Praxis und Referenz*, Markt und Technik, Buch- und Software-Verlag, München, 1999, ISBN 3-8272-5485-X.
- [68] Mustafa, H. Quresh, M. H. Samadzadeh: *Determining the Complexity of XML Documents Proceedings of the International Conference on Information Technology: Coding and Computing (ITCC'05)*, 2005.
- [69] Dr. Beatrice, Amrhein: *XML Schema Definition*, Jan 2010, Technik und Informatik, Berner Fachhochschule.
- [70] Jürgen, Bayer: *Nach Schema F-Grundlagen der XML-Validierung mit XML Schema 1.0*, April.2010.
- [71] Coy, Wolfgang: *Java & Webapplikationen*, SPC TEIA Lehrbuch-Verlag, Berlin, 2002, ISBN 3-935539-63-0.
- [72] Gumm, Heinz-Peter; Sommer, Manfred: *Einführung in die Informatik*, Oldenbourg Verlag, 8., vollst. überarb. Aufl., München, 2009, ISBN 978-3-486-58724-1.
- [73] Horstmann, Cay S.: *Core JAVA 2-1 Grundlagen*, Addison-Wesley Verlag, 4.,

- überarb. Fass., Übersetzung der 7. engl. Ausg., München, 2005, ISBN 3-8273-2216-2.
- [74] Holstege, Mary; Vedamuthu, Asir S.: *W3C XML Schema Definition Language (XSD): Component Designators*, W3C Working Draft, 2008.  
<<http://www.w3.org/TR/xmlschema-ref/>>;  
<<http://www.w3.org/TR/2008/WD-xmlschema-ref-20081117/>>.
- [75] Horstmann, Cay S.: *Core JAVA 2-2 Expertenwissen*, Prentice Hall Verlag, 7., überarb. Fass., Übersetzung der 7. engl. Ausg., München, 2005, ISBN 3-8273-2244-8.
- [76] Gao, Shudi (Sandy) 高殊镝; Sperberg-McQueen, Thompson, C. M.; Henry S.: *W3C XML Schema Definition Language (XSD) 1.1 Part 1: Structures*, 2009, W3C Working Draft.  
<<http://www.w3.org/TR/xmlschema11-1/>>;  
<<http://www.w3.org/TR/2009/WD-xmlschema11-1-20091203/>>
- [77] David, Peterson; Shudi (Sandy), Gao; 高殊镝; Ashok, Malhotra; C. M. Sperberg-McQueen; Henry S. Thompson : *W3C XML Schema Definition Language (XSD) 1.1 Part 2: Datatypes*, 2009.  
<<http://www.w3.org/TR/2009/WD-xmlschema11-2-20091203/>>
- [78] Uwe, Schneider; Dieter, Werner: *Taschenbuch der Informatik*, Hanser Fachbuch, Auflage: 5., April 2004.
- [79] Rahul, Srivastava: *XML Schema: Understanding Namespaces*, Oracle Technology Network, 2009.  
<[http://www.oracle.com/technology/pub/articles/srivastava\\_namespaces.html](http://www.oracle.com/technology/pub/articles/srivastava_namespaces.html)>
- [80] Toby, Baier; Michael, Ebert u.a.: *XML Schema Teil 0: Strukturen*, Mai, 2001.  
< <http://www.edition-w3c.de/TR/2001/REC-xmlschema-0-20010502/>>
- [81] John, Cowan; Richard, Tobin: *XML Information Set (Second Edition)*, 4. Feb 2004. W3C Recommendation, <<http://www.w3.org/TR/xml-infoset/>>



# Anhang

## A: Dokumentation von XSD

In diesem Abschnitt wird eine Übersicht über die für die vorliegende Diplomarbeit relevanten Dokumentationen des W3C gegeben. Die Angaben entsprechen dem Titel des jeweiligen Berichts und beinhalten das Veröffentlichungsdatum sowie eine kurze Beschreibung.

### XML-Schema – XML Schema Language

#### XML Schema 1.0

Die Spezifikation von XML Schema 1.0 besteht aus drei Dokumenten, von denen das erste eine Übersicht über den Standard geben soll, welcher im zweiten und dritten Teil beschrieben ist.

Titel : XML Schema Part 0: Primer Second Edition

Status: W3C Recommendation

seit : 28. Oct 2004

Hrsg.: David C. Fallside and Priscilla Walmsley

Links : <http://www.w3.org/TR/xmlschema-0/>  
<http://www.w3.org/TR/2004/REC-xmlschema-0-20041028/>

Titel : XML Schema Part 1: Structures Second Edition

Status: W3C Recommendation

seit : 28. Oct 2004

Hrsg.: Henry S. Thompson, David Beech, Murray Maloney and Noah Mendelsohn

Links : <http://www.w3.org/TR/xmlschema-1/>  
<http://www.w3.org/TR/2004/REC-xmlschema-1-20041028/>

Titel : XML Schema Part 2: Datatypes Second Edition

Status: W3C Recommendation

seit : 28. Oct 2004

Hrsg.: Paul V. Biron and Ashok Malhotra

Links : <http://www.w3.org/TR/xmlschema-2/>  
<http://www.w3.org/TR/2004/REC-xmlschema-2-20041028/>

## **XSD – XML Schema Definition Language**

### XSD 1.1

Die Spezifikation von XSD 1.1 besteht auch aus drei Dokumenten, von denen das erste eine zusammenfassende Einführung über den Entwurf geben soll, welcher im zweiten und dritten Teil beschrieben ist.

Titel : W3C XML Schema Definition Language (XSD): Component Designators

Status: W3C Working Draft

seit : 17. Nov 2008

Hrsg.: Mary Holstege and Asir S. Vedamuthu

Links : <http://www.w3.org/TR/xmlschema-ref/>  
<http://www.w3.org/TR/2008/WD-xmlschema-ref-20081117/>

Titel : W3C XML Schema Definition Language (XSD) 1.1 Part 1: Structures

Status: W3C Working Draft

seit : 3. Dez 2009

Hrsg.: Shudi (Sandy) Gao 高殊镝, C. M. Sperberg-McQueen and Henry S. Thompson

Links : <http://www.w3.org/TR/xmlschema11-1/>  
<http://www.w3.org/TR/2009/WD-xmlschema11-1-20091203/>

Titel: W3C XML Schema Definition Language (XSD) 1.1 Part 2: Datatypes

Status : W3C Working Draft

seit : 3. Dez 2009

Hrsg. : David Peterson, Shudi (Sandy) Gao 高殊镝, Ashok Malhotra,  
C. M. Sperberg-McQueen and Henry S. Thompson

Links : <http://www.w3.org/TR/xmlschema11-2/>  
<http://www.w3.org/TR/2009/WD-xmlschema11-2-20091203/>

## **SAWSDL – Semantic Annotations for WSDL and XML Schema**

Titel: Semantic Annotations for WSDL and XML Schema

Status : SAWSDL Recommendation

seit : 28. Aug 2007

Hrsg. : Joel Farrell and Holger Lausen

Links : <http://www.w3.org/TR/sawSDL/>  
<http://www.w3.org/TR/2007/REC-sawSDL-20070828/>

Titel: Semantic Annotations for WSDL and XML Schema — Usage Guide

Status : W3C Working Group Note

seit : 28. Aug 2007

Hrsg. : Rama Akkiraju and Brahmananda Sapkota

Links <http://www.w3.org/TR/sawSDL-guide/>  
<http://www.w3.org/TR/2007/NOTE-sawSDL-guide-20070828/>

Titel: SAWSDL Test Suite  
 Status : SAWSDL Recommendation  
 seit : 17. Jun 2007  
 Hrsg. : Jacek Kopecky Carine Bournez and Eric Prud'hommeaux  
 Links : <http://www.w3.org/2002/ws/sawSDL/CR/testsuite.html>

Titel: SAWSDL Candidate Recommendation Implementation Report  
 Status : SAWSDL Recommendation  
 seit : 26. Jun 2007  
 Hrsg. : Jacek Kopecky Carine Bournez and Eric Prud'hommeaux  
 Links : <http://www.w3.org/2002/ws/sawSDL/CR/>

## B: Zusätzliche Tabellen

Die Abbildung B1 zeigt anhand der JAXB Spezifikation die mögliche Umsetzung von XML Schema Datentypen in Java Datentypen.

Nr.	XSDDatentypen	Java Datentypen / Java Class
1	anySimpleType	java.lang.String/ java.lang.Object
2	anyAtomicType	java.lang.String/ java.lang.Object
3	anyURI	java.net.URI
4	base64Binary	byte[ ]
5	Boolean	Boolean
6	Date	javax.xml.datatype.XMLGregorianCalendar
7	dateTime	javax.xml.datatype.XMLGregorianCalendar
8	Decimal	java.math.BigDecimal
9	Double	Double
10	Duration	javax.xml.datatype.Duration
11	Float	Float
12	gDay	javax.xml.datatype.XMLGregorianCalendar
13	gMonth	javax.xml.datatype.XMLGregorianCalendar
14	gMonthDay	javax.xml.datatype.XMLGregorianCalendar
15	gYear	javax.xml.datatype.XMLGregorianCalendar
16	gYearMonth	javax.xml.datatype.XMLGregorianCalendar
17	hexBinay	byte[ ]
18	NOTATION	javax.xml.namespace.QName

19	precisionDecimal	javax.xml.datatype.XMLGregorianCalendar*
20	Qname	javax.xml.namespace.QName
21	String	java.lang.String
22	Time	javax.xml.datatype.XMLGregorianCalendar
23	dateTimeStamp	javax.xml.datatype.XMLGregorianCalendar*
24	dayTimeDuration	javax.xml.datatype.Duration
25	yearMonthDuration	javax.xml.datatype.Duration
26	Integer	java.math.BigInteger
27	nonPositiveInteger	java.math.BigInteger
28	negativeInteger	java.math.BigInteger
29	nonNegativeInteger	java.math.BigInteger
30	positiveInteger	java.math.BigInteger
31	unsignedLong	java.math.BigInteger*
32	unsignedInt	Long
33	unsignedShort	Int
34	unsignedByte	Short
35	Long	Long
36	int	Int
37	Short	Short
38	Byte	Byte
39	normalizedString	java.lang.String *
40	Name	javax.xml.namespace.QName*

Tabelle B1: Die Übersetzung von XSD Datentypen in Java Datentypen



Die Tabelle B2 listet teilweise die Abbildung der Implementierung von XSD4J über die Strukturen auf. Sie zeigt die Abbildung zwischen den Attributen und Elementen in XSD 1.1 und Java-Objekt.

Name der XSD 1.1	Attribute und Element	Java Objekt in XSD4J
alternative	id test type annotation simpleType complexType	String String QName XSDAnnotation() XSDType() XSDType()
Annotation	id appinfo documentation	String Sieh. Unter Appinfo Sieh. Unter Dokumentation
Appinfo	source any	String XSDAny()
Documentation	source any	String XSDAny()
Any	id maxOccurs minOccurs namespace processContents annotation	String Int Int String String XSDAnnotation()
Assert	id test annotation	String String XSDAnnotation()
Assertion	id test annotation	String String XSDAnnotation()
defaultAttributesApply	defaultAttributesApply	boolean
defaultOpenContent	appliesToEmpty id mode annotation any	Boolean String String XSDAnnotation() XSDAny()
Inheritable	-----	boolean
minScale	fixed id value annotation	Boolean String BigInteger XSDAnnotation()

maxScale	fixed id value annotation	Boolean String BigInteger XSDAnnotation()
openContent	Id Mode Annotation Any	String String XSDAnnotation() XSDAny()
override	Id schemaLocation annotation simpleType complexType group attributeGroup element attribute notation	String String XSDAnnotation() XSDType() XSDType() XSDSequence() XSDSequence() XSDElement() XSDAttribute() XSDAnnotation()
Ref	ref	QName

PS: ----- Die Attribute (kein Parameter)

Tabelle B2: Die Abbildung zwischen den neuen Strukturen  
in XSD 1.1 und Java Objekt

Die folgende Hardware- oder Software-Umgebung werden in dieser Arbeit verwendet:

Hardware-Umgebung	
Hersteller :	Acer
Modell:	Aspire 5738
Prozessor:	Pentium Dual-Core CPU T4200 @ 2.00GHz
Arbeitsspeicher (RAM):	4,00 GB
Systemtyp:	32 Bit-Betriebssystem
Software-Umgebung	
OS Name:	Windows Vista Home Premium, Service Pack 2
OS Version:	6.0
OS Architecture:	x86
Eclipse Platform:	Version: 3.4.2 Build id: M20090211-1700
Subversion:	svn, Version 1.6.6 (r40053) übersetzt Oct 26 2009, 20:14:36
Ant:	Apache Ant version 1.7.0 compiled on December 13 2006
Tomcat Version:	Apache Tomcat/6.0-snapshot
JVM Version:	Java(TM) SE Runtime Environment (build 1.6.0_16-b01)
JVM Vendor:	Sun Microsystems Inc.
java.version:	1.6.0_16

Tabelle B3: Hardware- oder Software-Umgebung

## C: Screenshots

Die Abbildung C1 zeigt die Oberfläche der Testfälle, die Einzelfall und mehrere Testfälle enthält. Die Testfälle können in XSD 1.0 oder XSD 1.1 überprüft werden.

Die Abbildung C2 enthält den Inhalt des überprüften XSD-Dokuments.


Die Abbildung C3 stellt die Ergebnisse von XSDParser und XSDDumper dar. Sie enthält ausführliche Informationen der Elemente und Attribute im geparsten XSD-Dokument.

Die Abbildung C4 zeigt die statistische Informationen von XSDParser und XSDDumper. Sie erstellt die Tabelle und Grafikdarstellung des überprüften XSD-Dokuments und enthält die Größe und Laufzeit des XSD-Dokuments.

Die Abbildung C5 stellt die Grafikdarstellung der Testfälle von Xerces-J im Kapitel 6 dar.

The screenshot displays the XSD4J web application interface. At the top right, there is a logo for XSD4J. The main heading reads "Diplomarbeit-- Vervollständigung der XML-Schema-Bibliothek XSD4J". Below this is a navigation menu with buttons for "Home", "Test", "XSD4J API", "Project Dynvocation", "TU Dresden Inf", "Note", and "About". The current time is shown as "Tue Apr 06 18:34:47 CEST 2010". The main content area is titled "Choice the XSD document" and contains a section for "parser from the website xsd document:" with a text input field containing "http://localhost:8080/XSD4JTest/test2\_11/all.xsd" and a "Parser Test The Website XSD Dokument" button. Below this, a section titled "Please, you can choice the following xsd documents:" lists "Einzele Testfälle" (Individual Test Cases) for "XSD 1.0" (with an "annotation" dropdown and "Abgeben" button) and "XSD 1.1" (with an "assertion" dropdown and "Abgeben" button). A section for "Mehrere Testfälle:" (Multiple Test Cases) lists "automatisch überprüfaren Testfällen für XSD 1.0" (with a "Parser Test" button) and "automatisch überprüfaren Testfällen für XSD 1.1" (with a "Selbst" dropdown and "Parser Test" button). At the bottom, it displays "Your IP address is 127.0.0.1 and your browser is Mozilla Firefox" and "copyright © 2009-2010".

Abbildung C1: Die Oberfläche der Testfälle



**Diplomarbeit--  
Vervollständigung der XML-Schema-Bibliothek XSD4J**

Time: Tue Apr 06 18:54:31 CEST 2010

[Home](#)
[Test](#)
[XSD4J API](#)
[Project Dynovocation](#)
[TU Dresden Inf](#)
[Note](#)
[About](#)

**XSD Dokument Name: assert.xsd und Parser**

---

**Parser this XSD: assert.xsd**

assert.xsd ▾ Quantity: 1

**This is original XSD: assert.xsd**

```
<?xml version="1.0" encoding="UTF-8"?>
<xs:schema xmlns:xs="http://www.w3.org/2001/XMLSchema">
  <xs:element name="Tag">
    <xs:complexType>
      <xs:sequence>
        <xs:element name="Vormittag" type="xs:time" />
        <xs:element name="Nachmittag" type="xs:time" />
      </xs:sequence>
      <xs:assert test="Vormittag gt Nachmittag" id="assert.id"/>
    </xs:complexType>
  </xs:element>

  <xs:element name="BookStore">
    <xs:complexType>
      <xs:sequence>
        <xs:element name="Book" maxOccurs="unbounded">
          <xs:complexType>
            <xs:sequence>
              <xs:element name="Title" type="xs:string" />
              <xs:element name="Author" type="xs:string" />
              <xs:element name="Date" type="xs:string" />
              <xs:element name="ISBN" type="xs:string" />
            </xs:sequence>
          </xs:complexType>
        </xs:element>
      </xs:sequence>
    </xs:complexType>
  </xs:element>
</xs:schema>
```

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

<xs:schema xmlns:xs="http://www.w3.org/2001/XMLSchema">

  <xs:element name="Tag">

    <xs:complexType>

      <xs:sequence> ... </xs:sequence>

      <xs:assert test="Vormittag gt Nachmittag" id="assert.id"/>

    </xs:complexType>

  </xs:element>

  <xs:element name="BookStore">

    <xs:complexType>

      <xs:sequence>

        <xs:element name="Book" maxOccurs="unbounded">

          <xs:complexType>

            ...

          </xs:complexType>

        </xs:element>

      </xs:sequence>

    </xs:complexType>

  </xs:element>
</xs:schema>
```

Abbildung C2: Der Inhalt des geparsen XSD-Dokuments

```

: Go back
! There are results of XSDParser and XSDDumper with all XSD documents:
BootstrappedSchema is:BootstrappedSchema11
===== TEST: parse all.xsd
===== TEST: parse http://127.0.0.1:8080/XSD4JTest/tests/all.xsd
-- parser successful!
<<org.dynvocation.lib.xsd4j.XSDParser>>: (xmlbase) Info: declaring prefix xsd for http://www.w3.org/2001/XMLSchema
<<org.dynvocation.lib.xsd4j.XSDParser>>: (xmlbase) Info: declaring prefix xsi for http://www.w3.org/1999/XMLSchema-instance
<<org.dynvocation.lib.xsd4j.XSDParser>>: (xmlbase) Info: declaring prefix xml for http://www.w3.org/XML/1998/namespace
<<org.dynvocation.lib.xsd4j.XSDParser>>: (xmlbase) Info: declaring prefix vc for http://www.w3.org/2007/XMLSchema-versioning
<<org.dynvocation.lib.xsd4j.XSDParser>>: Preloading schema with namespace null
<<org.dynvocation.lib.xsd4j.XSDParser>>: and 6 slices; level = 1
<<org.dynvocation.lib.xsd4j.XSDParser>>: Parsing XSD file http://127.0.0.1:8080/XSD4JTest/tests/all.xsd
<<org.dynvocation.lib.xsd4j.XSDParser>>: Parsing all XSD files and elements
<<org.dynvocation.lib.xsd4j.XSDParser>>: ~~~~~
<<org.dynvocation.lib.xsd4j.XSDParser>>: Preloading in effect
<<org.dynvocation.lib.xsd4j.XSDParser>>: - 1 files plus 0 preloads
<<org.dynvocation.lib.xsd4j.XSDParser>>: - 0 elements plus 0 preloads
<<org.dynvocation.lib.xsd4j.XSDParser>>: - 0 schemas plus 1 preloads
<<org.dynvocation.lib.xsd4j.XSDParser>>: Multiparser parser level: 1
<<org.dynvocation.lib.xsd4j.XSDParser>>: -- ***** --
<<org.dynvocation.lib.xsd4j.XSDParser>>: -- multi-schema file parser begin
<<org.dynvocation.lib.xsd4j.XSDParser>>: -- parse additional file http://127.0.0.1:8080/XSD4JTest/tests/all.xsd

: Go back
! There are results of XSDParser and XSDDumper:
BootstrappedSchema is:BootstrappedSchema10
===== TEST: parse attributes.xsd
===== TEST: parse http://127.0.0.1:8080/XSD4JTest/tests/attributes.xsd
-- parser successful!
<<org.dynvocation.lib.xsd4j.XSDParser>>: (xmlbase) Info: declaring prefix xsd for http://www.w3.org/2001/XMLSchema
<<org.dynvocation.lib.xsd4j.XSDParser>>: (xmlbase) Info: declaring prefix xsi for http://www.w3.org/1999/XMLSchema-instance
<<org.dynvocation.lib.xsd4j.XSDParser>>: (xmlbase) Info: declaring prefix xml for http://www.w3.org/XML/1998/namespace
<<org.dynvocation.lib.xsd4j.XSDParser>>: (xmlbase) Info: declaring prefix vc for http://www.w3.org/2007/XMLSchema-versioning
<<org.dynvocation.lib.xsd4j.XSDParser>>: Preloading schema with namespace null
<<org.dynvocation.lib.xsd4j.XSDParser>>: and 6 slices; level = 1
<<org.dynvocation.lib.xsd4j.XSDParser>>: Parsing XSD file http://127.0.0.1:8080/XSD4JTest/tests/attributes.xsd
<<org.dynvocation.lib.xsd4j.XSDParser>>: Parsing all XSD files and elements
<<org.dynvocation.lib.xsd4j.XSDParser>>: ~~~~~
<<org.dynvocation.lib.xsd4j.XSDParser>>: Preloading in effect
<<org.dynvocation.lib.xsd4j.XSDParser>>: - 1 files plus 0 preloads
<<org.dynvocation.lib.xsd4j.XSDParser>>: - 0 elements plus 0 preloads
<<org.dynvocation.lib.xsd4j.XSDParser>>: - 0 schemas plus 1 preloads
<<org.dynvocation.lib.xsd4j.XSDParser>>: Multiparser parser level: 1
<<org.dynvocation.lib.xsd4j.XSDParser>>: -- ***** --
<<org.dynvocation.lib.xsd4j.XSDParser>>: -- multi-schema file parser begin
<<org.dynvocation.lib.xsd4j.XSDParser>>: -- parse additional file http://127.0.0.1:8080/XSD4JTest/tests/attributes.xsd
<<org.dynvocation.lib.xsd4j.XSDParser>>: -> loading file http://127.0.0.1:8080/XSD4JTest/tests/attributes.xsd

===== TEST: dumper
ok!
<xsd:schema xmlns:pfx='test' xmlns:xsd='http://www.w3.org/2001/XMLSchema' xmlns:xsi='http://www.w3.org/1999/XMLSchema-instance' xmlns:vc='http://www.w3.org/2007/XMLSchema-versioning' targetNamespace='test'>
<xsd:element name='root'>
<xsd:complexType>
<xsd:sequence>
<xsd:element name='elDayTimeDuration' type='pfx:dtdDerived' />
</xsd:sequence>
</xsd:complexType>
</xsd:element>
<xsd:simpleType name='dtdBase'>
<xsd:restriction base='xsd:dayTimeDuration'>
<xsd:minInclusive value='P1D' />
<xsd:maxInclusive value='P2DT2H' />
</xsd:restriction>
</xsd:simpleType>
<xsd:simpleType name='dtdDerived'>
<xsd:restriction base='pfx:dtdBase'>
<xsd:minInclusive value='P51H' />
<xsd:maxInclusive value='P100H' />
</xsd:restriction>
</xsd:simpleType>
</xsd:schema>
-- All tests passed!

Run time: 44 ms

```

Abbildung C3: Die Ergebnisse von XSDParser und XSDDumper

~~~~~XSWildcardTest01.xsd~~~~~is end!~~~~~

All Run time: 1550 ms

Die Folgende ist die Tabelle der XSDs und Ausführungszeit:

| XSD Name        | a.xsd | address.xsd | base.xsd | idc.xsd | ipo.xsd | otherNamespace.xsd | personal.xsd | schema_imp.xsd | tests.xsd | xmlbase.xsd |
|-----------------|-------|-------------|----------|---------|---------|--------------------|--------------|----------------|-----------|-------------|
| Groesse(Bytes): | 163   | 1692        | 2412     | 1200    | 1881    | 489                | 2218         | 409            | 1751      | 410         |
| Zeit:           | 28.0  | 48.0        | 79.0     | 46.0    | 97.0    | 39.0               | 53.0         | 39.0           | 95.0      | 48.0        |

Die Folgende ist das Histogramm der Beziehungen ueber XSDs und Ausführungszeit:

Das Histogramm browst in das separaten Fenster: >>[Links](#)<<

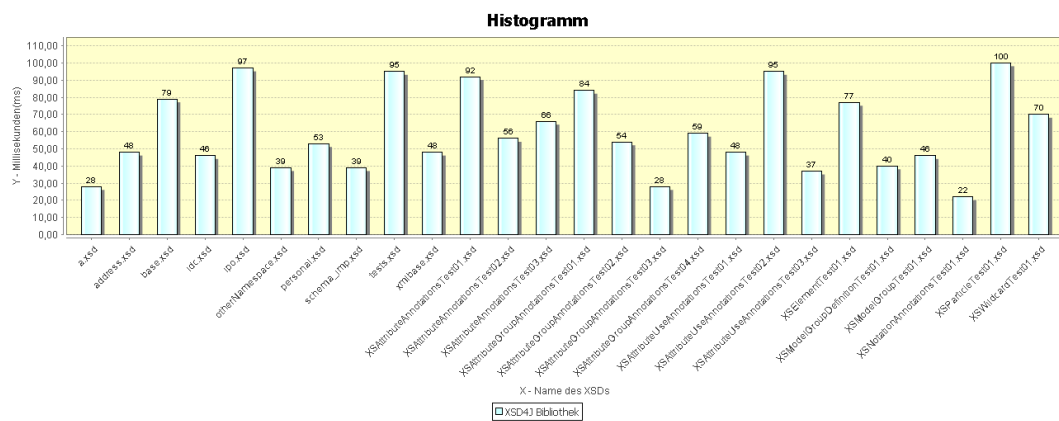


Abbildung C4: Die statistische Informationen von XSDParser und XSDDumper

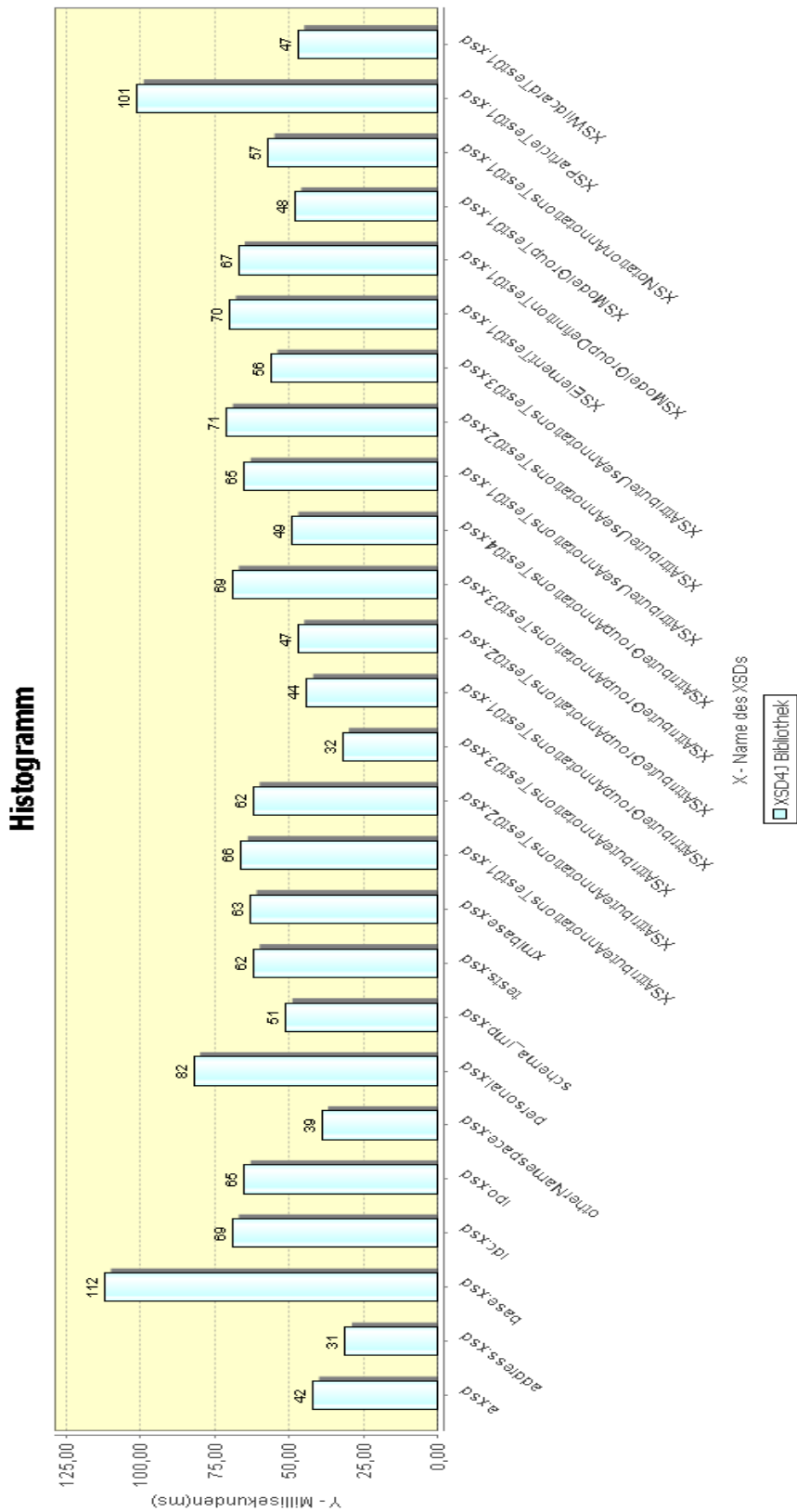


Abbildung C5: Die Ergebnisse der Testfälle von Xerces-J (Sehen kapitel 6 )



## D. Index der Abbildungen, Tabellen und Listings

### A. Auflistungen der Abbildungen:

Abbildung 2.1: Überblick über die Sprachfamilie XML  
Abbildung 3.1: Die Beziehung zwischen Instanz und Schema  
Abbildung 3.2: Die Komponenten eines XML-Schemas  
Abbildung 3.3: Hierarchie der Datentypen  
Abbildung 4.1: Die Architektur der XSD4J Bibliothek  
Abbildung 4.2: XSDParser für das Element Annotation  
Abbildung 5.1: Beziehung zwischen XSD 1.0 und XSD 1.1  
Abbildung 5.2: Beziehung des Validators zwischen XSD 1.0 und XSD 1.1  
Abbildung 5.3: Die Struktur von Kommunikation  
Abbildung 5.4: Datentypen der Spezifikation XSD 1.1  
Abbildung 5.5: Die abgeleitete Baumstruktur der Datentypen  
Abbildung 6.1: Konzept der Testfälle  
Abbildung 6.2: MVC Entwurfs Muster in XSD4JTest  
Abbildung 6.3: Die Komponenten von MVC Entwurfs Muster in XSD4JTest  
Abbildung 6.4: Die Struktur von XSD4JTest  
Abbildung 6.5: Gruppe Test von W3C  
Abbildung 6.6: Gruppe Test von Saxon EE (20 mal)  
Abbildung 6.7: Gruppe Test von Saxon EE (100 mal)  
Abbildung 7.1: SAWSDL im Bild  
Abbildung 7.2: Lifting und Lowering

### B. Auflistungen der Tabellen:

Tabelle 2.1: Vergleich des XML-Schemas mit DTD  
Tabelle 3.1: Die verschiedenen Arten in den Facetten  
Tabelle 3.2: Die Beschreibung der Facetten  
Tabelle 4.1: Die Bibliotheken, Klassen und Dokumente in der XSD4J Bibliothek  
Tabelle 4.2: Die Packages in der XSD4J Bibliothek  
Tabelle 4.3: Die Methoden in XSDParser  
Tabelle 4.4: Die Methoden in XSDDumper  
Tabelle 5.1: Die Anwendungen der verfügbaren neuen Strukturen und Facetten  
in die Elemente oder Attribute von XSD 1.1  
Tabelle 5.2: Die Attribute in Namensraumversion  
Tabelle 5.3: Die erweiterten Strukturen in XSD 1.1  
Tabelle 5.4: Die neuen Strukturen in XSD 1.1  
Tabelle 5.5: Die Strukturen der neuen Facetten in XSD 1.1

Tabelle 5.6: Überblick nach der vervollständigten Implementierung von XSD4J

Tabelle 6.1: Die Information der Testfälle

Tabelle 6.2: Die URLs der Testfälle

Tabelle 6.3: Die Ergebnisse der Testfälle in der Implementierung

Tabelle 6.4: Die Ergebnisse der Testfälle von W3C

Tabelle 6.5: Die Ergebnisse der Testfälle von Xerces-J

Tabelle 6.6: Die Ergebnisse der Testfälle von Saxon EE

Tabelle 6.7: APIs in Package xsd4japp

Tabelle 6.8: APIs in Package Applet

Tabelle 6.9: Information der Servlets

Tabelle 6.10: Information der JSPs

Tabelle 6.11: Information des Ordners

Tabelle 7.1: Die Erweiterung der Elemente oder Attribute von XSD4J

### **C. Auflistungen der Listings:**

Listing 2.2: Die Darstellungen der verschiedenen Namensräume im XML-Schema

Listing 3.3.1.1: Beispiel für komplexe Elemente

Listing 3.3.1.2: Beispiel für einfache Elemente

Listing 3.3.2.1: Beispiel für Attribute

Listing 3.3.3.1: Beispiel für globale und lokale Deklarationen

Listing 3.3.4.1: Beispiel für All

Listing 3.3.4.2: Beispiel für falsche All

Listing 3.3.4.3: Beispiel für Choice

Listing 3.3.4.4: Beispiel für verschachtelte Gruppen

Listing 3.3.5.1: Beispiel für minOccurs-Attribut und maxOccurs-Attribut

Listing 3.3.6.1: Beispiel für include

Listing 3.3.6.2: Beispiel für redefine

Listing 3.3.6.3: Beispiel für import

Listing 3.3.7: Beispiel für Identifizierungseinschränkung

Listing 3.3.8.1: Beispiel für Elementgruppen

Listing 3.3.8.2: Beispiel für Attributgruppen

Listing 3.3.9.1: Beispiel für Vereinigungen

Listing 3.3.9.2: Beispiel für Vereinigungen

Listing 3.3.10: Beispiel für Kommentare

Listing 3.4.5.1: Beispiel für Einschränkungen

Listing 3.4.5.2: Beispiel für Listen

Listing 3.4.5.3: Beispiel für den Vereinigungs-Datentyp

Listing 3.4.6: Die Facette

Listing 4.2.2.1: annotation.xsd

Listing 4.2.2.2: Die Struktur des Elements Annotation

Listing 4.2.2.3: Aufruf der Methode parseXSDAnnotation in der Klasse XSDParser

- Listing 4.2.2.4: Die Methode `parseXSDAnnotation` in der Klasse `XSDParser`
- Listing 4.2.3.1: Aufruf der Methode `xsdDump_annotation` in der Klasse `XSDDumper`
- Listing 4.2.3.2: Die Methode `xsdDumper_annotation` in der Klasse `XSDDumper`
- Listing 5.2.1: Ein XML-Fragment mit den Namensräumen
- Listing 5.2.2: `schema.xsd`
- Listing 5.2.4: `assert.xsd`
- Listing 5.2.5.1: `alternative.xsd`
- Listing 5.2.5.2: `kommunikationtype-alternative.xsd`
- Listing 5.2.5.3: `kommunikationtype-inneren-alternative.xsd`
- Listing 5.2.6: `publictiontype-assert.xsd`
- Listing 5.2.7: `defaultAttributes.xsd`
- Listing 5.2.8.1: `openContent.xsd`
- Listing 5.2.8.2: `openContent.xsd`
- Listing 5.2.8.3: `openContent.xsd`
- Listing 5.2.8.4: `defaultOpenContent.xsd`
- Listing 5.2.9.1: `all-1.xsd`
- Listing 5.2.9.2: `all-2.xsd`
- Listing 5.3.1: `assertion.xsd`
- Listing 5.3.2: `explicitTimezone.xsd`
- Listing 5.3.3: `maxScale.xsd`
- Listing 7.3.1: Erweiterung für das Attribut `modelReference`
- Listing 7.3.2: Erweiterung für das Attribut `loweringSchemaMapping`