Web-basierte Anwendungen 2: verteilte Systeme
Dokumentation Phase I

David Ferdinand Petersen
11083571

10.04.13

Inhaltsverzeichnis

1 Einleitung

In dieser Dokumentation befinden sich die ausgearbeiteten Inhalte der Phase 1 des Moduls Web-basierte Anwendungen 2: verteilte Systeme der ersten Projektphase. Die Modellierungs- und Codeaufgaben befinden sich im Java-Projekt des Repositorys, die schriftlichen Ausarbeitungen sind in dieser Dokumentation zu finden.

2 Hauptteil

Im Folgenden sind Lösungen und Verweise zu den Aufgaben zu finden.

2.1 Begriffserklärung im Bezug auf XML und XML-Schema (Aufgabe 1)

2.1.1 Wohlgeformtheit

Der Begriff 'Wohlgeformtheit' beschreibt XML-Dokumente mit gültiger Syntax. Hierbei gilt, folgende Hinweise zu beachten[1, Seite 35][2]:

- Zu jedem XML-Dokument gehört genau ein Wurzel-Element
- Alle Elemente müssen zwingend einen schließenden Tag besitzen
- Bei allen Tags wird zwischen Groß- und Kleinschreibung unterschieden
- Elemente müssen richtig verschachtelt werden
- Attribute der Elemente müssen zwischen Anführungsstrichen (bzw. Apostroph) stehen
- Alle Zeichen müssen dem festgelegten Zeichensatz entsprechen

2.1.2 Validität

Validität bezeichnet ein XML-Dokument, welches gegen ein DTD oder XML-Schema[1] geprüft und als valide eingestuft wird und zudem Wohlgeformtheit aufweist. DTD steht in diesem Falle für 'Document Type Definition' und definiert die Struktur eines XML-Dokuments, in welchem erlaubte Elemente und Attribute vorgegeben sind.[2]

2.1.3 Namespace

Ein Namespace (deutsch: 'Namensraum') bietet die Funktion Konflikte aufgrund von gleicher Elementnamen zu vermeiden. Ein solcher Konflikt kann beim Zusammenführen von mehreren XML-Dokumenten auftreten, da eventuell gleiche Elementnamen vergeben worden sind, welche eine unterschiedliche Bedeutung

[1] Siehe 2.3.3 auf Seite 3 für weitere Informationen zu XML-Schema.

aufweisen. Indem ein Namensraum inklusive Präfix angegeben wird, können diese Konflikte vermieden werden.[3]

2.2 Schlagzeugweltrekord XML-Dokument (Aufgabe 2)

Die XML- und JSON-Dokumente für die Aufgaben befinden sich im Git-Repository[2], können jedoch auch direkt in dem Browser aufgerufen werden[3]:

2.2.1 a) XML-Dokument

Das XML-Dokument (Dateiname: Aufgabe2a.xml), welches sich auf die gegebene Webseite für den Schlagzeugweltrekord bezieht, befindet sich unter folgendem Link:

https://github.com/Cr4ckX/wba2_ss13/blob/master/Java Phase 1/Aufgabe2/Aufgabe2a.xml

2.2.2 b) JSON-Dokument

Das aus dem XML-Dokument abgeleitete JSON-Dokument (Dateiname: Aufgabe2b.json), welches sich somit ebenfalls auf die gegebene Webseite für den Schlagzeugweltrekord bezieht, befindet sich unter folgendem Link:

https://github.com/Cr4ckX/wba2_ss13/blob/master/Java Phase 1/Aufgabe2/Aufgabe2b.json

2.3 Rezept XML-Dokument (Aufgabe 3)

2.3.1 a) Erstellen eines XML-Dokuments

Zu dieser Aufgabe wurden zwei Lösungsvorschläge geliefert. Diese lassen sich ebenfalls, wie auch die Dateien aus Aufgabe 2, im Git-Repository finden. Die beiden Dateien unterscheiden sich lediglich von der Struktur, der Inhalt bleibt weitestgehend gleich.

Das Dokument Aufgabe3a_1[4], im Ordner Aufgabe 3, enthält alle Informationen des Rezeptes in *Elementen* geschachtelt. Das Dokument Aufgabe3a_2[5], ebenfalls im Ordner Aufgabe 3, hingegen, enthält zusätzlich *Attribute*, welche Informationen des Rezepts darstellen.

In XML ist es nicht fest definiert, das heißt, es gibt keinerlei Regeln, welche bestimmen, welche Lösung als richtig oder falsch angesehen werden darf. Allgemein sind beide Möglichkeiten zulässig, jedoch beschreiben Attribute (laut W3Schools.org[4]) meist zusätzliche Informationen zu einem Element, welche

[2] https://github.com/Cr4ckX/wba2_ss13/tree/master/Java Phase 1/src

[3] Jedoch ist die Darstellung, direkt im Browser unvorteilhaft: Die Struktur des Codes verschiebt sich leider.

[4] Direkter Link: https://github.com/Cr4ckX/wba2_ss13/blob/master/Java Phase 1/Aufgabe3/Aufgabe3a_1.xml

[5] Direkter Link: https://github.com/Cr4ckX/wba2_ss13/blob/master/Java Phase 1/Aufgabe3/Aufgabe3a_2.xml

nicht direkt Teil der Daten sind, sondern z.B. für die Weiterverarbeitung wichtig erscheinen. Als Beispiel sei das Attribut

```
id='1'
```

genannt. Je nach Verwendung, kann dieses Attribut nun für eine eindeutige Zuordnung des Elementes verhelfen. Da die eindeutige Zuordnung jedoch nicht Teil des eigentlichen Rezeptes ist, da es für das Rezept keine relevanten Daten - wie z.B. Zutaten - enthält, wird diese Zuordnungsinformation meist als Attribut abgespeichert. Des Weiteren seien Attribute schwer zu lesen und zu handhaben[4]. Aus diesem Grund, wird das Dokument *Aufgabe3_a)_1* bevorzugt und als primären Lösungsvorschlag eingestuft.

2.3.2 b) Gemeinsamkeiten der Rezepte

Die Struktur der Rezepte bleibt stets gleich. Ein Rezept gliedert sich demnach aus Zutaten und deren Zubereitung. Die einzelnen Informationen, also konkret benötigte Zutaten und Zubereitungsinformationen variieren jedoch von Rezept zu Rezept. Natürlich ändert sich je Rezept die Bewertung, (evtl.) der Autor und die Kommentare. Im Allgemeinen ist die Maginalspalte je Rezept flexibel, welche jedoch nicht in der XML-Datei betrachtet werden soll und daher an dieser Stelle nicht näher beleuchtet wird.

2.3.3 c) Kriterien für ein XML-Schema

Bevor die Kriterien für ein XML-Schema für das bestehende XML-Dokument ausgearbeitet werden, soll zunächst einmal verdeutlicht werden, worum es sich eigentlich bei einem XML-Schema handelt.

Ein XML-Schema hat die selbe Aufgabe, wie eine Document Type Definition, nämlich die Struktur eines XML-Dokumentes spezifizieren. Jedoch mit einigen Unterschieden. Beispielsweise ist das XML-Schema selber auch in XML geschrieben. Allgemein wird behauptet, ein XML-Schema sei mächtiger als eine DTD[5, Seite 29][8]. Beide Versionen haben ihre Vor- und Nachteile. Beispielhafte Unterschiede sind:

- Mit einer DTD lassen sich eigene Entities definieren, mit XML-Schemas nicht.
- XML Schemas ermöglichen es, Datentypen zuzuweisen und gar eigene Datentypen zu erstellen.
- XML Schemas unterstützen Namespace, DTDs nicht.
- ...

Für eine weitergehende, detaillierte Beschreibung bezüglich des XML-Schemas soll auf das W3C[6] verwiesen werden, da an dieser Stelle nur einleitend der Begriff 'XML-Schema' erläutert werden sollte.

[6] http://www.w3.org/TR/xmlschema-0/

Für die Ausarbeitung relevanter Kriterien wurden die gegebenen Fragestellungen zunächst abgearbeitet.

Welche Daten müssen in simple und welche in complex-types abgebildet werden? Simple-Types sind Elemente (und Attribute), welche nur Text, also entsprechend keine weiteren Kindelemente oder Attribute enthalten. Complex-Types sind somit alle Elemente, welche Kindelemente und/oder Attribute besitzen. Das Attribut selbst ist immer ein simple-type.

Im folgenden wird nun das aus Abschnitt 2.3.1 auf Seite 2 erstellte Dokument 'Aufgabe3a_1' in Betracht gezogen. Hier ist nun ersichtlich, dass alle Elemente, welche weitere Elemente oder Attribute enthalten als complex-types abgebildet werden müssen. Darunter fallen entsprechend folgende Elemente:

Rezepte Enthält Kindelemente.

Rezept Enthält Attribut und Kindelemente.

Zutaten Enthält Kindelemente.

Zutat Enthält Kindelemente.

Zubereitung Enthält Kindelemente.

Brennwert Enthält Kindelemente.

Kommentare Enthält Kindelemente

Kommentar Enthält Attribut und Kindelemente.

Datum Enthält Kindelemente.

Uhrzeit Enthält Kindelemente.

Entsprechend sind alle übrigbleibenden Elemente simple-types. Sie dürfen wie bereits beschrieben keine weiteren Elemente oder Attribute enthalten. Folgende simple-type Elemente sind im Dokument 'Aufgabe3a_1' enthalten:

id ist Attribut von Rezept/Kommentar.

Rezptname ist Kindelement von Rezept.

Name ist Kindelement von Zutat.

Menge ist Kindelement von Zutat.

Einheit ist Kindelement von Zutat.

Personen ist Kindelement von Brennwert.

Anzahl ist Kindelement von Brennwert.

Einheit ist Kindelement von Brennwert.

Beschreibung ist Kindelement von Zubereitung.

Nutzer ist Kindelement von Kommentar.

Tag ist Kindelement von Datum.

Monat ist Kindelement von Datum.

Jahr ist Kindelement von Datum.

Stunde ist Kindelement von Uhrzeit.

Minute ist Kindelement von Uhrzeit.

Inhalt ist Kindelement von Kommentar.

Für welche Daten ist die Abbildung in Attributen sinnvoller? Diese Frage wurde bereits in Abschnitt 2.3.1 auf Seite 2 beantwortet. Es wurde mit dem Lösungsvorschlag verblieben, in Attributen lediglich Werte zu speichern, welche sich nicht auf die eigentlich zu speichernden Informationen beziehen. Entsprechend wurde das Dokument 'Aufgabe3a_1' angepasst. Jedoch besteht immer noch die Möglichkeit Attribute bei Bedarf einzufügen:

Zum Beispiel könnte man bei der Zutat mit dem Namen 'Eier' anstelle der Einheit "default", ein Attribut hinzufügen, welches dem Parser/Programm klar macht, dass es für 'Eier' keine Einheit gibt, bzw. keine angezeigt wird. Ein anderes Beispiel: Bei dem Element 'Brennwert' bestehen Kindelemente, welche 'Personen', 'Anzahl' und 'Einheit' heißen. Hier besteht nicht nur Verwechslungsgefahr mit dem Element 'Einheit' als Kindelement des Elements 'Zutat', welche mittels eines Attributs verhindert werden könnte, auch die Personenanzahl, auf die sich der Wert des Brennwertes bezieht, ist meist *immer* für nur eine Person bestimmt. So können lediglich Abweichungen (wie zwei oder mehr Personen) als Attribut vermerkt werden. Damit ist gemeint, dass, sollte diese Angabe sich auf mehrere Personen beziehen, und nicht auf den default-Wert 'eine Person', könnte man sich das Element 'Person' sparen und dies lediglich in einem Attribut unterbringen.

Der Einfachheit und Übersichtlichkeit halber wurde zunächst jedoch die bereits bestehende Variante verwendet. Spätestens bei der Entwicklung des Programmes, welches sich auf die XML und XML-Schema Dateien bezieht, wird sich die Tauglichkeit der vorgestellten XML-Rezept-Struktur, hinsichtlich der Attribute, herausstellen.

Welche Datentypen müssen für die Elemente definiert werden? Welche Restriktionen müssen definiert werden? Prinzipiell müssen keine eigenen Datentypen für dieses Rezept definiert werden. Lediglich für die Elemente

'Einheit' (Kindelement von Zutat/Arbeitszeit/Brennwert) könnte ein selbst definierter Datentyp erstellt werden, damit sichergestellt wird, dass keine willkürliche Einheiten verwendet werden. Des Weiteren könnte das Element 'Schwierigkeitsgrad' vordefinierte Presets anbieten, sodass auch hier ein selbst definierter Datentyp in Frage käme. Der Grund, warum gleich zwei Fragestellungen hier abgearbeitet werden, ist, dass Datentypen generell bereits Restriktionen mit sich bringen. Es handelt sich nämlich bereits um eine Restriktion, wenn 'km/h' nicht als 'Einheit' verwendet werden kann. Im Folgenden sind Vorschläge für Datentypen (D) und Restriktionen (R) der einzelnen Elemente und Attribute gelistet:

(Rezept-)id Attribut von Rezept. D: integer, R: obligatorisch, inkrementell, eindeutig

Rezeptname D: string, R: obligatorisch

Zutaten R: mindestens eine Zutat, keine maximale Anzahl Zutaten

- **Name** D: string
- **Menge** D: integer, R: nicht kleiner oder gleich 0
- **Einheit** D: string/selbst_definierter_Datentyp_Zutaten

Zubereitung

- **Arbeitszeit** R: optional
  - **Zeit** D: integer/number, R: nicht kleiner oder gleich 0
  - **Einheit** D: string/selbst_definierter_Datentyp_Arbeitszeit
- **Brennwert** R: optional
  - **Personen** D: integer, R: nicht kleiner oder gleich 0
  - **Anzahl** D: integer/number, R: nicht kleiner oder gleich 0
  - **Einheit** D: string/selbst_definierter_Datentyp_Brennwert
- **Beschreibung** R: obligatorisch

Nachträglich: Kommentare

- **Kommentar** R: optional
- **(Kommentar-)id** Attribut von Kommentar. D:integer, R: obligatorisch, inkrementell, eindeutig
- **Nutzer** D: string, R: optional
- **Datum, Uhrzeit** D: dateTime/selbst_definierter_Datentyp mit Restriktionen wie (R): gültige Werte (z.B. 25 Uhr wäre ungültig!)

Restriktionen, wie das Begrenzen von maximalen Zeichen oder Ziffern, bzw. der möglich verwendbaren Zeichen oder Ziffern wurde hier aufgrund der mangelnden Notwendigkeit nicht berücksichtigt.[7] 'White Space'-Restriktionen wurden ebenfalls nicht angesprochen, da der Default-Wert bereits 'preserve' beträgt, was besagt, dass Leerzeichen nicht entfernt werden.

Dass ein Element obligatorisch ist, muss nicht explizit angegeben werden, da jedes Element (sofern nicht explizit angegeben) standardmäßig einen Default Wert bei der minimalen Anzahl Vorkommnisse (minOccurs) von 1 hat. Entsprechend müssen die maximalen Vorkommnisse (maxOccurs) bei den Elementen auf 'unbounded' gesetzt werden, welche mehrere[8] Vorkommnisse erlauben. Diese Elemente sind zum Beispiel:

- *Rezept* - es gibt evtl. mehrere Rezepte
- *Zutaten* - zu einem Rezept gibt es mindestens eine Zutat (minOccurs auf default belassen), evtl. mehrere
- oder das Element *Kommentar* - es können keine (minOccurs = 0) oder mehrere Kommentare abgegeben werden.

Ein Attribut wird als obligatorisch deklariert, sobald der das Attribut in der Schema-Datei den Parameter 'use=required' enthält.

Nachtrag: Für ein abgegebenen Kommentar (Element wurde leider erst nachträglich hinzugefügt) wurden Restriktionen bei dem eigenen Datentyp für Datum sowie der Uhrzeit vorgenommen, sodass richtige Daten und Uhrzeiten angegeben werden müssen[9].

Weitere Kriterien Es besteht die Möglichkeit, mehrere Elemente als *global* zu definieren. Das erste Element (Wurzelelement) der XML-Datei, also im Schema, das Kindelement von <xs:schema> hat einen globalen Ansprechbereich. Das heißt so viel wie, dass unabhängig wo das Element sich in welcher Verschachtelung befindet, es überall in der Schema-Datei angesprochen werden darf. Weitere Elemente, also nicht nur das Wurzelelement als global zu definieren, ist besonders Sinnig, wenn mehrere Schema-Dateien vorliegen, bzw. mehrere XML-Dateien mit nur *einem* Schema geprüft werden sollen. [5, Seite 29ff]

Verwendung von Any-Elementen: Mit einem any-Element ließe sich in einer Schema-Datei ein 'Wildcard-Element' hinzufügen. Diese Elemente sind nicht fest in der Schema-Datei definiert, es wird jedoch durch das Any-Element die Möglichkeit geboten, zusätzliche Elemente, welche nicht im Schema festgelegt sind, in die XML-Datei einzubauen (erhöhte Flexibilität aber schlechtere Handhabung). Das heißt, das XML-Dokument wäre gegen eine Schema-Datei mit Any-Elementen (möglicherweise) auch valide, sollten Elemente vorkommen, welche

[7] Nachträglich für das Datum und die Uhrzeit eines Kommentars jedoch doch noch verwendet.

[8] bzw. durch 'unbounded' sogar unendlich viele

[9] Alternativ: xs:dateTime Datentyp verwenden, dann sind diese Restriktionen nicht mehr notwendig. Jedoch zeigte dieses Beispiel immerhin das Be- bzw. Eingrenzen von möglichen zu verwendenden Ziffern.

nicht in der Schema-Datei festgelegt wurden. Diese Any-Elemente haben einen sinnvollen Einsatz, sobald weitere Informationen zu dem Elternelement hinzugefügt werden sollen, welche zunächst nicht vorgesehen waren. In dem gegebenen Fall jedoch, soll eine feste Struktur mit der Schema-Datei festgelegt werden, weshalb keinerlei Any-Elemente vorkommen werden.[6]

Schließlich besteht noch die Möglichkeit Elementnamen in einer Schema-Datei mit anderen Elementnamen zu substituieren. Damit ist gemeint, dass der Elementname1 mit dem Elementnamen2 substituiert werden kann, ohne, dass dies die Validität des XML-Dokumentes beeinträchtigt. Sinnvoll ist dies, um die XML-Datei für mehrere Sprachen zugänglich zu machen und entsprechender Elementnamen-Anpassung zu ermöglichen. Auch die Möglichkeit der Substitution soll in der Schema-Datei nicht beachtet werden, da der Nutzen dieser Funktion für die anzufertigende Schema-Datei nicht erkennbar ist.[7]

2.3.4 d) Erstellen des XML-Schemas

Das XML-Schema befindet sich, wie die anderen Dokumenten aus Aufgabe 3 in dem Ordner 'Aufgabe 3' im Git-Repository. Es wurden zur Lösung der Aufgabe wieder *zwei* Dokumente angefertigt. Das erste Dokument zeigt den ersten Modellierungsversuch eines XML-Schemas[10]. Dort ist deutlich zu erkennen, dass bei erhöhter Komplexität des Schemas die sog. 'Russian Doll'-Darstellung [5, Seite 30] nicht unbedingt gut geeignet ist. Aus diesem Grunde wurde ein weiteres Dokument[11] mit der selben Semantik angefertigt, welche jedoch eine leicht abgeänderte Syntax aufweist. Diese Abwandlung dient lediglich der Übersichtlichkeit halber; beide Dokumente können trotzdem gleichwertig verwendet werden.[12]

Die Vorschläge für die Datentypen und Restriktionen wurden weitestgehend aus der vorherigen Aufgabe übernommen. Es sei noch erwähnt, dass die Möglichkeit der selbst definierten Einheiten für Zutat, Arbeitszeit und Brennwert zwar verwendet werden könnten, jedoch sind möglicherweise nicht alle Einheiten bekannt und demnach nur eine begrenzte, evtl. unvollständige 'Auswahl' vorhanden, sodass diese Restriktion allenfalls Nachteile mit sich bringt. Es wurden diese Restriktionen jedoch beispielhaft bei der Arbeitszeit festgelegt, jedoch nicht fest als verwendeter Datentyp implementiert. Das Element 'Schwierigkeitsgrad' hat ebenfalls einen eigenen Datentyp erhalten (dieses Mal incl. Zuordnung), sodass die Implementierung und Darstellungsweise eigener Datentypen wenigstens einmal veranschaulicht werden können.

Die XML-Datei, aus Aufgabe 3d) (Dateiname: Aufgabe3d.xml), welche gegen das erstellte Schema geprüft wird, ist natürlich wohlgeformt und kann erfolgreich gegen beide gelieferten XML-Schemas geprüft und als valide eingestuft werden.

[10] Direkter Link: https://github.com/Cr4ckX/wba2_ss13/blob/master/Java Phase 1/Aufgabe3/Aufgabe3d_1.xsd

[11] Direkter Link: https://github.com/Cr4ckX/wba2_ss13/blob/master/Java Phase 1/Aufgabe3/Aufgabe3d_2.xsd

[12] Ersichtlich ist, dass dieses "aufgeteilte Schema" zwar übersichtlicher ist, aber auch Element- und Attributname eindeutig identifizierbar sein sollten. Allgemein wäre es besser jedes Element und Attribut eindeutig zu benennen. (Z.B.: id -> rezeptId, id -> kommentarId)

2.4 XML-Programm (Aufgabe 4)

2.4.1 a) Erzeugen von Java Objekten

Genaugenommen wurden Klassen erzeugt. Diese wurden mittels Terminal erzeugt, da das Plugin Schwierigkeiten mit sich brachte. Dies sollte jedoch nicht weiter einschränken.

2.4.2 b) Programm erstellen

Testweise wurde die Lösung nach und nach erarbeitet. Das Programm wurde jedoch nach der Erstellung der Lösung für Aufgabe b) nochmals modifiziert und in zwei Klassen aufgeteilt, damit es für die folgende Aufgabe übersichtlich bleibt.

2.4.3 c) Einfügen von Kommentaren

Auch hier wäre es, wie bei dem XML-Schema, von Vorteil gewesen, zunächst die Aufgabenstellung genau zu interpretieren und auch zu diskutieren, was mit dem Begriff 'Kommentare' gemeint ist. Zunächst wurde davon ausgegangen, dass es sich um XML-Kommentare der Form

```
<!-- Kommentar -->
```

handelt (siehe Git-Versionshistorie). Doch nach auftretenden Hartnäckigkeiten kam Zweifel auf, woraufhin ein Mitarbeiter um die genaue Bedeutung befragt wurde. Es handle sich bei 'Kommentaren' um Rezept-Kommentare, welche von Nutzern zu einem entsprechenden Rezept abgegeben werden können. Diese waren zunächst nicht in der XML-Datei berücksichtigt, wurden dann jedoch entsprechend im Nachhinein angepasst. Die anfänglichen Schwierigkeiten wurden also nicht nur dank der schnellen Antwort des Mitarbeiters gelöst, auch die zunächst als 'unwichtig' eingestuften Informationen innerhalb der XML-Datien wurden glücklicherweise nachträglich ergänzt und die XML-Datei sowie das XML-Schema nochmals angepasst.

Das Programm selber funktioniert mit switch-cases. Das heißt, das geforderte Menü, wird mit dem Einlesen von numerischen Eingaben realisiert. Wird demnach gefordert, die XML-Datei in einer lesbaren Form auszugeben, so muss in die Konsole eine "1" eingegeben werden, und diese Auswahl mit Enter bestätigt werden. Wird die 2 ausgewählt, wird die Möglichkeit geboten, der XML Datei Rezept-Kommentare hinzuzufügen. Dazu wird zunächst gefragt, welchem Rezept ein neuer Kommentar hinzugefügt werden soll. Mittels dem sog. Marshalling (Java Objekte -> XML) wird der hinzugefügt Kommentar in der festgelegten Datei gespeichert.

2.4.4 Reflexion - XML im Programm verarbeiten

Im Allgemeinen stellte sich heraus, dass die verwendete Struktur der XML-Datei so, wie sie bereits unter 2.3.3 auf Seite 5 versuchsweise entwickelt wurde, bereits

sehr gut funktioniert. Alle wichtigen Informationen sind in Elementen enthalten, Attribute enthalten lediglich Informationen zur Verarbeitung der Daten. So wird beispielsweise die Rezept-ID - gespeichert in einem Attribut - mehrmals für die Zuordnung, aber auch Zählung der Rezepte verwendet. Nachträglich, bei dem Element 'Kommentare', erwies sich jedoch, dass Uhrzeiten und Daten in entsprechend vorgefertigten Datentypen (xs:dateTime) oder in Attributen eventuell besser handzuhaben sind.

Natürlich könnte das Programm weiter optimiert werden, wie z.B. Behebung fehlerhafter Benutzereingaben oder Wiederholen der case Abfrage. Im Fokus steht jedoch die Arbeit mit einer XML-Datei/Schema, aus diesem Grunde sind diese extra Funktionen nicht implementiert, können aber optional nachträglich hinzugefügt werden.

2.5 XML vs. JSON (Aufgabe 5)

Formate wie XML oder JSON bieten die Möglichkeit, Textdaten zu einer selbst definierten Struktur zu speichern. Dies ist in sofern sinnvoll, da diese Formate sehr simpel aufgebaut sind und entsprechend der Umgang der Daten einfacher ist, als beispielsweise mit einem komplexem Datenbankschema. Die Aufgabe 4 hat gezeigt, wie simpel es sein kann, die selbst definierte Struktur eines XML-Dokumentes in einem Programm zu verarbeiten, wie es in einer Datenbank nicht ohne weiteres möglich wäre. Ersichtlich ist, dass die Flexibilität von XML einen großen Vorteil hat, jedoch gibt es auch festgelegte Strukturen, wie ein XML-Dokument auszusehen hat. Siehe dafür Namespaces und genormte Schemata.

XML ist für Menschen leicht lesbar, erweiterbar, simpel, flexibel und weit verbreitet. Jedoch ist XML aber auch komplizierter als JSON. XML muss aufwendig geparst werden, hat eine komplexe Syntax im Gegensatz zu JSON - was für eine Markuplanguage auch sehr sinnvoll ist - jedoch für den reinen Austausch von Daten („exchange format“) ist JSON besser geeignet. JSON hat eine kompakte, für den Austausch von Daten reduzierte Syntax, muss nicht geparst werden, da sie aus JavaScript besteht und ist ebenfalls leicht lesbar für den Menschen, jedoch ist JSON nicht erweiterbar und nicht so verbreitet wie XML.

Allgemein kann man sagen, dass JSON unkomplizierter, kleiner und einfacher ist als XML, dafür jedoch nicht die über Komplexität für eine Auszeichnungssprache verfügt.[9, 10, 11, 12]

3 Fazit

Trotz keinerlei Vorkenntnisse oder früherer Erfahrung mit diesem Dateiformat, erwiesen sich die Aufgaben als lösbar und es war interessant, diese auszuarbeiten. Zusammenfassend lässt sich der Schluss ziehen, dass dieses Projekt gezeigt hat, wie wichtig XML für das Web, aber auch, wie flexibel und simpel diese erweiterbare Auszeichnungssprache ist.

Literatur

[1] Prof. Dr. K. Fischer: Foliensatz K1, K2 - Web-basierte Anwendungen 2: Verteilte Systeme - Sommersemester 2013

[2] http://www.w3schools.com/xml/xml_dtd.asp (Aufgerufen am 18.03.13)

[3] http://www.w3schools.com/xml/xml_namespaces.asp (Aufgerufen am 18.03.13)

[4] http://www.w3schools.com/xml/xml_attributes.asp (Aufgerufen am 18.03.13)

[5] Sas Jacobs: Beginning XML with DOM and Ajax. From Novice to Professional. New York: Springer-Verlag 2006

[6] http://www.w3schools.com/schema/schema_complex_any.asp (Aufgerufen am 01.04.13)

[7] http://www.w3schools.com/schema/schema_complex_subst.asp (Aufgerufen am 01.04.13)

[8] http://www.w3schools.com/schema/schema_why.asp (Aufgerufen am 01.04.13)

[9] http://www.w3schools.com/json/

[10] http://www.json.org/xml.html

[11] http://blog.doubleslash.de/mit-json-auf-der-uberholspur/

[12] http://de.softuses.com/73650