WBA2 SS13 - Phase 1 Ausarbeitung

Ngoc-Anh Dang 11082691

Abgabe am 12.04.2013

Inhaltsverzeichnis

Teil I
Aufgaben

1 Begriffsklärung: Wohlgeformtheit, Validität, Namespaces

Erklären Sie kurz die Begriffe Wohlgeformtheit, Validität und Namespaces im Bezug auf XML und XML-Schema.

Wohlgeformtheit Wenn ein XML-Dokument wohlgeformt ist, bedeutet das, dass das XML-Dokument „lesbar“ ist. Das XML-Dokument muss dazu eine korrekte Baumstruktur aufweisen. Konkrete Bedingungen dafür sind beispielweise:

- je XML-Dokument darf nur 1 Wurzelknoten existieren
- ein Element eines XML-Dokuments mit Inhalt muss zwischen einem öffnenden und schließenden Tag sein
- außer das Wurzelelement muss jedes Element von Elementen vollständig umschlossen sein
- ein Attributwert muss zwischen ' oder " stehen
- die in dem XML-Dokument verwendeten Zeichen müssen dem festgelegten Zeichensatz entsprechen

Validität Wenn ein XML-Schema valid ist, bedeutet das, dass die Syntax gegenüber den formulierten Festlegungen korrekt ist. Man kann zum einen ein XML-Schema selbst validieren und zum anderen ein XML-Dokument gegen ein XML-Schema validieren. Im letzteren Fall prüft man, ob diese einander entsprechen.

Namespaces Namespaces oder auf deutsch „Namensräume“ dienen zur Festlegung des Vokabulars eines XML-Dokuments. Dies bietet einem die Möglichkeit mehrere Namensräume zu verwenden, ohne, dass es Uneindeutigkeiten z.B. bei gleichen Elementnamen gibt (Verhinderung von sog. „clashing names“, also Unterstützung von sublanguages). Die Namensräume werden eindeutig durch eine URI identifiziert und durch das Attribut `xmls` beschrieben. I.d.R. werden die Namensräume zu Anfang einmal mit Präfixen (z.B. `xmls:xsd`) deklariert, welche dann im folgendem Schema die Element-, Typ-Namen, etc. eindeutig bestimmen. Der default namespace lautet `xmls="";` der für XML-Schema ist `xmlns="http://www.w3.org/2001/XMLSchema"`.

[1][2][3]

2 Äquivalenz zwischen XML und JSON

2.1 XML

Erzeugen Sie ein XML-Dokument, dass die Daten des folgenden Formulars vollständig erfasst: `http://www.gm.fh-koeln.de/~vsch/anmeldung/gruppenanmeldung.html` **. Füllen Sie das Dokument mit einem Beispieldatensatz. Achten Sie darauf, dass über das Formular mehrere Personen gleichzeitig erfasst werden können. Wichtig: Es sollte nicht die HTML-Struktur der Webseite in der XML-Datei abgebildet werden, sondern die zu übertragenden Daten.**

Lösung `xml/aufgabe2a.xml`

In dieser Lösung lautet der Wurzelknoten `Anmeldungen`. Dieser kann mehrere Kindknoten vom Typ `Gruppe` haben, welche eindeutig anhand der `gruppen_id` identifizierbar sind. Desweiteren hat jede Gruppe genau einen Gruppenleiter, welcher als Attribut im Gruppenelement angegeben wird. Das Gruppenelement hat wiederum Kindelemente, welche die Daten der Mitglieder der Gruppe erfassen. Jedes Mitglied stellt einen Knoten dar. Die persönlichern Daten und die Informationen zum Schlagzeug, sowie eine Mitglieds-ID werden in dieser Lösung als Attribute gespeichert. Die Mitglieds-ID des Gruppenleiters wird im Attribut `gruppenleiter` des Gruppenelements gespeichert.

Zu beachten ist, dass im Gegensatz du den restlichen Attributen die ID-Attribute nicht vom Nutzer eingegeben werden, sondern automatisch vergeben werden. Dies könnte für Verwirrung sorgen. Eine Möglichkeit das zu umgehen ist ein Attribut wie `rolle` einzuführen, in welcher der Wert `gruppenleiter` oder `mitglied` gespeichert wird. Da diese Verwirrung jedoch die Teilnehmer nicht betrifft, sondern eventuell nur Programmierer, erscheint mir dieses Contra-Argument nicht zureichend um meinen gewählten Lösungsweg zu ändern.

Desweiteren kann in diesem XML-Dokument nicht sichergestellt werden, dass eine Gruppe mindestens 2 Mitglieder haben muss. Dies kann man in einem zugehörigem XML-Schema festlegen.

Eine weitere alternatives Lösungskonzept ist, die vielen Attribute als Kindelemente darzustellen. Ich habe mich gegen diesen Lösungsweg entschieden, da zum Einen die Attribute, meines Erachtens, eine noch übeschaubare Menge sind und zum Anderen, ich diese eher als Eigenschaften eines Mitglieds interpretiere, anstatt als „Bestandteile“ (wie es bei einer Darstellung als Kindknoten interpretieren würde). Noch ein Argument gegen die Elementdarstellung ist, dass diese Variante Zeilen spart.

2.2 JSON

Erzeugen Sie ein JSON-Dokument, dass zu ihrem XML-Dokument äquivalent ist.

Lösung `xml/aufgabe2b.json`

Diese Lösung ist komplett am XML-Dokument der Teilaufgabe a) orientiert und behinhaltet nur eine kleine Änderung: Unter dem Attribut `schlagzeug` wird kein String mehr gespeichert (`‘‘vorhanden‘‘`/ `‘‘nicht vorhanden‘‘`), sondern boolean-Werte (`true`/ `false`). In XML gibt es ebenfalls den Datentyp boolean, dieses Wissen habe mir im Zeitpunkt der Bearbeitung der Aufgabe jedoch noch nicht erarbeitet. Diese Variante mit den Wahrheitswerten ist effizienter, da sie „kürzer“ sind als Strings. D.h. z.B. muss auf diese Weise im Schema keine `enumeration` programmiert werden und spart somit Komplexität ein. Bis auf diesen Aspekt sind beide Dokumente äquivalent. [4][5][6]

3 Rezepte in XML

3.1 XML-Dokument zu Rezept

Gegeben ist folgendes Rezept: `http://www.chefkoch.de/rezepte/24641006006067/Lenchen-s-Schokoladenkuchen.h`
Entwickeln Sie ein XML-Dokument, in dem die Daten des Rezeptes abgebildet werden. Achten Sie darauf, dass das Dokument semantisch möglichst reichhaltig ist. Bei dieser und den folgenden Aufgaben lassen sie bitte die Daten in der Marginalspalte auf der rechten Seite weg.

Lösung `xml/aufgabe3a.xml`

Bei der Erstellung dieses XML-Dokuments bin ich ähnlich wie beim XML-Dokument der Nr. 2a vorgegangen. Ich habe mehr mit Attributen gearbeitet, anstatt mit Kindelementen (`id`, `portionen`, `schwierigkeit`, ...), wieder mitderselben Argumentation. Außerdem lassen sich die Werte dieser Attribute relativ kurz darstellen (ein Wort oder eine Zahl). Die Elemente `bilder`, `zutaten`, `schritte` und `kommentare` dahingegen besitzen, meines Erachtens, eine etwas komplexere Struktur und sind in wiederum semantisch sinnvolle Unterelemente unterteilbar.

Als „Bruch“ dieser Vorgehensweise könnte man meine Umsetzung der Zubereitungsschritte ansehen: Hier habe ich jeden einzelnen Schritt als ein eigenes Element dargestellt, anstatt als einen Fließtext. Diese Modularisierung erschien mir sinnvoll, da ich mehrere Szenarien kenne, in denen ein Prozess - als welchen man eine Zubereitung ansehen kann - step-by-step dargestellt wird. Als Beispiel kann man einen Screenreader nennen, der die Schritte vorliest und zwischen diesen Pausen einlegt. Auf diese Weise können Sehbehinderte dem Rezept besser folgen, als wenn der Screenreader diesen in einem Zug vorliest.

3.2 Gemeinsamkeiten und Unterschiede

Betrachten Sie nun andere Rezepte auf der Webseite `http://www.chefkoch.de`**. Beschreiben Sie welche Gemeinsamkeiten die Rezepte hinsichtlich ihrer Daten haben und worin Sie sich unterscheiden.**

Gemeinsamkeiten Daten, welche in jedem Rezept vorkommen sind Titel des Rezepts, Angabe der Zutaten und dessen Menge und die Angabe der Zubereitungsschritte.

Unterschiede Daten, die nicht immer in jedem Rezept auftauchen sind diverse Einheiten („nach Belieben“, „Spritzer“, „Dose“) und neue Eigenschaften, wie „Ruhezeit“, „Koch-/Backzeit“. Manche Angaben wie „Brennwert“ oder die „Portionen“ fehlen bei anderen Rezepten. Zudem hat nicht jedes Rezept hat ein Bild. Überdies gibt es Unterschiede im Kommentar-Abschnitt: Nicht jedes Rezept hat Kommentare oder manche Kommentare enthalten Antwort-Kommentare.

3.3 Kriterien für ein XML-Schema

Arbeiten Sie die Kriterien heraus, die für die Entwicklung einer XML-Schema-Datei beachtet werden müssen. Die Schema-Datei soll die Struktur für eine XML-Datei definieren, in der mehrere unterschiedliche Rezepte gespeichert werden können. Ziel ist es, dass das XML-Schema möglichst restriktiv ist, so dass in der XML-Datei möglichst semantisch sinnvolle Daten bezüglich der Rezepte gespeichert werden können. Ziehen Sie beim Aufstellen der Kriterien u.A. folgende Fragestellungen in Betracht: Welche Daten müssen in simple und welche in complextypes abgebildet werden? Für welche Daten ist die Abbildung in Attributen sinnvoller? Welche Datentypen müssen für die Elemente definiert werden? Welche Restriktionen müssen definiert werden?

Bei der Entwicklung einer XML-Schema-Datei muss man wissen, was als Element und was als Attribut definiert werden soll. Nach weiteren Recherchen und entgegen meiner bisherigen Vorgehensweise, ist es bei einem Schema zu vermeiden Attribute zu verwenden [1]. Ein Grund dafür ist, dass die Struktur der Daten verloren geht bzw. stark vereinfacht wird und damit die Lesbarkeit verloren gehen kann. Desweiteren können Attribute keine geordnete Werte enthalten und sind schwer zu Erweitern, im Gegensatz zu Elementen, zu welchen in einer bestimmten Reihenfolge Unterelemente hinzugefügt werden können. Attribute eignen sich für Werte, die nur einmal im Dokument vorkommen, beispielsweise Identifikatoren. Außerdem die Primärfunktion

von XML der Datenaustausch, daher sollten alle Informationen, welche deutlich als Daten wahrgenommen werden, als Elemente definiert werden.

Ein weiteres Kriterium für die Entwicklung einer XML-Schema-Datei ist, dass man sich ein gutes Bild der Struktur machen soll, damit man weiß, welche Elemente welche und wieviele Kindelemente haben. Überdies kann es auch von Bedeutung sein, in welcher Reihenfolge die Elemente und Attribute beschrieben werden (`<sequence>`/ `<all>`). Eine weitere Überlegung ist ob man ein Element als „leer" also nur mit Attributen definiert, oder mit Text zwischen den Tags definiert. Darüberhinaus sollten die Datentypen (simple-/ complex-type) der Elemente und Attribute bekannt sein. Dazu ist es sinnvoll sich sich überlegen, ob es default-Werte, feste Werte oder Wertebereiche angegeben werden müssen.

Aufgrund diesen neuen Erkenntnissen, weichen meine Antworten teilweise von dem XML-Dokument der Teilaugfabe a) ab.

Welche Daten müssen in Simple- und welche in Complex-Types abgebildet werden und welche Datentypen müssen für die Elemente definiert werden?

SimpleType

anyURI `bild_url`

string `user, zutatname, schritt, kommentar`

positiveInteger `stunde, minuten, brennwert`

decimal `menge`

dateTime `datum/uhrzeit`

ComplexType `chefkochbuch, bilder, rezept, zutaten, zutat, schritte, kommentare, back-/koch-zeit`

Für welche Daten ist die Abbildung in Attributen sinnvoller?

In diesem Fall sehe ich nur die ID als sinnvolles Attribut an, da dies die einzige Information ist, welche nicht zwingend als Information für den Leser relevant ist und man sie zu den Metadaten zählen kann.

Welche Restriktionen müssen definiert werden?

- Bei den Mengenangaben der Zutaten dürfen nur gültige Einheiten verwendet werden.
- Bei den Schwierigkeitsstufen darf nur zwischen leicht, mittel und schwer gewählt werden.
- Die Angabe der Minuten dürfen nur zwischen 0 und 59 sein.
- Manche Angaben sind optional (`ruhezeit, bilder, kommentare, brennwert`), andere verpflichtend (`menge, einheit, portionen, id, rezeptname, zutatname`).
- Es sollte mindestens ein Rezept, eine Zutat und ein Schritt geben.

3.4 Erstellung des XML-Schema

Erstellen Sie nun ein XML-Schema auf Basis ihrer zuvor definierten Kriterien. Generieren Sie nun auf Basis des Schemas eine XML-Datei und füllen Sie diese mit zwei unterschiedlichen und validen Datensätzen.

Lösung `xml/aufgabe3d-schema.xsd`

Lösung `xml/aufgabe3d-data.xsd`

Von der Aufgabe 3.4 und der gesamten Aufgabe 4 (die ja auf das Schema der 3.4 aufbaut) ist diese die 2. Version. Die alte Version (zu finden unter `Phase1/old`) ist ebenfalls beinah voll funktionsfähig (lediglich das marshalling fehlt), dennoch habe ich mich dazu entschieden eine nochmal komplett ein neues Schema zu entwickeln. Ein Grund ist, dass ich mehr und besser auf meine Ergebnisse aus den vorherigen Aufgaben

aufbauen möchte. Auf der einen Seite habe ich das Schema inhaltlich ausgebaut: Nun können mehr Informationen gespeichert werden, etwa Bilder zum Rezept, User, welche diese und die Kommentare posten können, die Uhrzeit, wann ein Kommentar beigetragen wurde und einige „Attribute“ eines Rezept, wie der Brennwert, die Kochzeit etc. Strukturell bin ich ebenfall anders vorangegangen: Anstatt für jedes Element einen eigenen Typ zu definieren (globale Deklaration), habe ich alle lokal und verschachtelt definiert. Auf diese Weise reduziere ich einerseits die Anzahl der aus dem xjc kompilierten Java-Klassen und somit auch die Kompexität des Programmmes und andererseits, werden nur sinnvolle neue complex-types definiert (das war ein Tipp der Tutoren). Desweiteren wird das Schema dadurch übersichtlicher.

In dem Testdatensatz habe ich 2 Rezepte eingefügt. Das Erste enthält alle möglichen Informationen, beim Zweiten entfallen manche optionalen Elemente. Somit konnte ich direkt prüfen, ob die Optionalität funktioniert.

4 Entwicklung des JAVA-Programms

4.1 Java-Okjekte Erzeugung mittels des xjc

Erzeugen Sie zunächst aus der Schema-Datei der vorherigen Aufgabe Java-Objekte. Nutzen Sie dazu den XJC-Befehl über das Terminal und fügen Sie die generierten Klassen ihrem Java-Projekt hinzu. Alternativ zur Terminal-Eingabe existiert ein JAXB Eclipse Plug-In.

Lösung `Phase1/src`

4.2 Entwicklung des Rezepte-Programms

Entwickeln Sie nun das Java-Programm. Es soll die XML-Datei öffnen, einlesen und die enthaltenen Daten über die Konsole wieder ausgeben. Benutzen Sie bitte bei der Bearbeitung der Aufgabe die generierten JAXB-Klassen aus der vorherigen Teilaufgabe.

Lösung `Phase1/src`

Die Kommentarfunktion der folgenden Aufgabe habe ich direkt in den Code dieser Aufgabe implementiert, weswegen ich leider nicht direkt die Lösung dieser Aufgabe vorzeigen kann. Alle erarbeiteten Funktionen dieser Aufgabe sind jedoch unverändert in die nächste Aufgabe übernommen worden.

4.3 Hinzufügen der Kommentarfunktion

Erweitern Sie ihr Programm so, dass es möglich ist, über die Konsole neue Kommentare zu einem Rezept hinzuzufügen. Benutzen Sie auch hierfür die generierten JAXB-Klassen. Erstellen Sie ein Menü, das in der Konsole angezeigt wird. Über dieses Menü sollen die Auswahl der Funktionen, zum Ausgeben der Daten und Erstellen neuer Kommentare, möglich sein.

Lösung `Phase1/src`

5 Vergleich: XML und JSON

Diskutieren Sie, warum es sinnvoll ist Daten in Formaten wie XML oder JSON zu speichern. Stellen Sie außerdem die beiden Formate gegenüber und erläutern Sie kurz deren Vor- und Nachteile.

	XML	JSON
Einfachheit	Öffnende und schließende Tags	Syntax ist einfacher gestaltet -> lesbarer, leichter schreibbar, Overhead Reduktion Keine Tags, nur key + value
Komplexität	Werte und Eigenschaften können sowohl als Attribute oder als Kindknoten beschrieben werden -> kann zu Problemen führen	In JSON kann dieses Problem nicht auftreten.
Typisierung	Man kann zusätzlich eigene Typen definieren.	JSON-Daten sind im Gegensatz zu XML-Daten typisiert, wobei nur einige grundlegende Typen unterstützt werden.
		Eine JSON-Definition in JavaScript direkt mit der eval()-Funktion in ein JavaScript-Objekt umsetzen (JSON ist valides JavaScript)
Einsetzbarkeit	XML ist eine Auszeichnungssprache -> vielseitiger einsetzbar	JSON ist ein Datenaustauschformat -> weniger vielseitig einsetzbar
Verbreitung	XML ist weiter verbreitet	JSON verdrängt XML aufgrund seiner Einfachheit dort, wo keine komplizierten Auszeichnungen notwendig sind

Fazit Je nach Komplexität der Datenstruktur kann man entscheiden, ob man Daten in einem XML- oder JSON-Dokument speichert. Bei vielen Daten und Daten mit einer komplexen Struktur ist XML empfehlenswerter, da man dank mehr Möglichkeiten in der Sprache präziser arbeiten kann. Handelt es sich um einen einfachen Datenaustausch eignet sich JSON mehr. Bei XML ist das Datenvolumen im Gesamten mehr, da die Daten der Tags mit übertragen müssen, wobei bei JSON lediglich der key mit dem value übertragen werden muss. Dieser Unterschied ist jedoch nicht allzu gravierend, da dank der hohen Bandbreiten heutzutage die Zeitdifferenz kaum spürbar ist. Im Allgemeinen ist es nützlich Daten in Formaten wie JSON oder XML zu speichern, da die Daten strukturiert werden und so schneller und gezielter verarbeitet werden können, beispielsweise im Vergleich zu Datenbanken.

[4][5][6]

Teil II
Fehlerbehandlung und Ausblick

6 Probleme während der Entwicklung des Programms

Das Größte Problem, mit welchem ich mich während der Programmierung befasst habe ist folgendes: Das Programm wirft eine Null-Pointer-Exception, wenn man einen Kommentar zum 2. Rezept hinzufügen will - vorausgesetzt, es wurde vorher noch nicht ausgegeben. Wenn es zuvor aufgerufen wurde, wird keine Exception geworfen. Dieser Konflikt tritt beim 1. Rezept nicht auf. Anhand des Debuggers lässt sich feststellen, dass das Kommentar-Objekt samt der eingegeben Daten noch fehlerfrei erstellt werden kann. Beim Hinzufügen des Kommentar-Objekts zum Rezept tritt die besagte Exception dann auf (`rezept.kommentare.kommentar.add(K);`). Durch die Ausgabe `if(rezept.kommentare==null)System.out.print("aha.");` bzw. `if(rezept.kommentare.kommentar==null)System.out.print("aha.");` habe ich herausgefunden, dass das `kommentar`-Objekt null ist. Nun habe ich eine Bedingung implementiert, welche nun jetzt ein kommentar-Listenobjekt erstellt und dem Rezept hinzufügt, falls `rezept.kommentare.kommentar==null`. Somit ist das Programm voll funktionsfähig.

7 Ausblick

Während der Entwicklung sind mir natürlich weitere verbesserungsfähige Aspekte aufgefallen am Programm aufgefallen, wobei aufgrund des Zeitmangels mir jedoch nicht mehr gelungen ist diese zu bearbeiten. Darunter ist zu nennen, dass die Einheiten der Mengen noch nicht vollständig sind. Außerdem ist es angenehmer, wenn man nicht unbedingt die Reihenfolge einhalten muss wenn man die allgemeinen Infos in die XML-Datei eingibt. Dies ist mit `<all>` realisierbar. Ein weiterer Punkt ist, dass die Uhrzeit ist noch sehr kryptisch dargestellt wird. Mit einem Formatter ist es möglich die Ausgabe lesbarer zu machen. Zudem wird bei der Kochzeit „`null` Stunden“ ausgegeben, wenn die Arbeitszeit weniger als eine Stunde dauert. Dies kann man dadurch lösen, dass man bereits im XML-Schema einen default-Wert zuweist. Unschön ist es außerdem, dass direkt nach einer Ausgabe das Menü ebenfalls wieder aufgerufen wird. Das führt dazu, dass die Zeilen der Rezeptausgabe hochrutschen, sodass der Leser gezwungen ist zu scrollen. Als Lösung kann man eine kürzere Zwischenabfrage implementieren, in welcher der Nutzer durch eine Eingabe zurück zum Menü gelangt, etwa „mit Enter kommen Sie zurück zum Menue“. Die Möglichkeit Kommentare zu Kommentaren abzugeben, wie ich es in der 3.2 erarbeitet habe, habe ich ebenfalls nicht geschafft umzusetzen.

Teil III
Lernunterlagen und Quellen

[1] http://www.w3schools.com/schema/ , http://www.w3schools.com/xml/default.asp (letzter Zugriff: 10.04.2013)

[2] http://www.teialehrbuch.de/Kostenlose-Kurse/XML/7743-Attribute-oder-Elemente.html (Zugriff: 08.04.2013)

[3] „An Introduction to XML and Web Technologies“, Anders Moller and Michael Schwartzbach, Addixon Wesley, 2006

[4] http://www.json.org/xml.html (Zugriff: 06.04.2013)

[5] http://de.wikipedia.org/wiki/JavaScript_Object_Notation (Zugriff: 06.04.2013)

[6] http://predic8.de/xml-json-yaml.htm (Zugriff: 06.04.2013)