



Fachhochschule Köln
Cologne University of Applied Sciences

WBA2 – Web-basierte Anwendungen 2:

Verteilte Systeme

Prof. Dr. Kristian Fischer
Workshop SoSe 2013

Projektdokumentation Phase 2

Sinem Kaya (11084460)

Selda Akyol (11075378)

Inhalt

- 1 Vorwort
- 2 Aufgaben – Phase 1
 - 2.1 Aufgabe 1
 - 2.2 Aufgabe 2
 - 2.3 Aufgabe 3
 - 2.4 Aufgabe 4
 - 2.5 Aufgabe 5
- 3 Aufgaben – Phase 2
 - 3.1 Aufgabe 1 XML - Schema
 - 3.2 Aufgabe 2 Ressourcen und Semantik der HTTP-Operationen
 - 3.3 Aufgabe 3 RESTful Webservice
 - 3.4 Aufgabe 4 Konzeption + XMPP Server einrichten
 - 3.5 Aufgabe 5 XMPP - Client
 - 3.6 Aufgabe 6 Client – Entwicklung
- 4 Fazit
- 5 Quellenangabe

1. Vorwort

In der ersten Phase des Workshops zur Veranstaltung Web-basierte Anwendungen 2: verteilte Systeme haben wir uns mit der Modellierung von Daten, welche zur Kommunikation von Systemkomponenten benötigt werden, unter Verwendung von XML und XML-Schema, beschäftigt.

Da wir einige Fehler bzw. Lücken in unseren Aufgaben hatten, haben wir uns dafür entschieden gemeinsam die Aufgaben der ersten Phase nochmal zu überarbeiten, was dazu führt, dass wir ein gemeinsames Dokument abgeben.

Nach Abschluss der ersten Phase haben wir uns anschließend mit dem Erstellen von Ressourcen etc. beschäftigt.

StudSNewsSource - Projektbeschreibung

Unser Projekt basiert auf ein Anwendungssystem, die es in erster Linie Studenten/innen ermöglicht intern Daten in Bezug auf Ihr Fachgebiet auszutauschen bzw. anderen Kommilitonen zur Verfügung zu stellen.

Das Anwendungssystem besteht aus drei Hauptteilern: "Modul", "Professoren", "News".

In dem Reiter "Module" wird den Studenten die Gelegenheit zur Verfügung gestellt, individuelle Prioritäten per Abonnements der aufgelisteten Module zu setzen, wobei noch die Möglichkeit besteht mehrere Module aber auch Professoren zu abonnieren.

Die Abonnements stellen in unserem Fall die asynchrone Kommunikation unseres Systems dar, .

Neben den angebotenen Modulen werden Informationen dargestellt wie Modulbeschreibungen, Semester, Professoren, Prüfungsart, Voraussetzungen und Dateien zum Downloaden.

Hier findet unsere synchrone Kommunikation statt in dem die Daten ausgetauscht werden.

Unter anderem wird dem Anwender die Möglichkeit gegeben, aktuelle News in Verbindung mit der zutreffenden Hochschule zu gewährleisten, was im Reiter "News" aufrufbar ist.

2.1

Aufgabe 1

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

Namespaces

XML-Namensräume bieten eine einfache Möglichkeit, um Element- und Attributnamen, die in "Extensible Markup Language"-Dokumenten verwendet werden können, eindeutig zu benennen. Die Element- und Attributnamen werden mit Namensräumen verknüpft, die durch URI-Verweise identifiziert werden. Außerdem werden Namespaces dazu benutzt um in einem XML Dokument mehrere unterschiedliche XML-Sprachen zu mischen.

Wohlgeformtheit

(well-formed documents) bedeutet das sich ein Dokument an die Regeln des W3C (World Wide Web Consortium) halten muss. Das W3C wurde gegründet um das volle Potenzial des Webs auszuschöpfen.

Dazu werden einheitliche Spezifikationen, Richtlinien, Software und Tools entwickelt, die denn Fortschritt des Webs fördern. Ein XML Dokument muss in einem einzelnen Wurzel Element eingeschlossen sein.

Alle unbedingt erforderlichen Attribute sind angegeben, befinden sich im richtigen Wertebereich und entsprechen dem angegebenen Typ. Für die eingesetzten Elemente gilt, dass diese korrekt ineinander verschachtelt sind.

Außerdem besitzt jedes XML-Dokument nur ein Wurzelement und Nicht-Wurzel-Elemente müssen von überliegenden Elementen umschlossen werden.

Elemente mit Inhalten besitzen ein öffnendes und ein schließendes Tag und Attributwerte werden von Anführungsstrichen eingeschlossen.

Ein XML-Dokument ist dann wohlgeformt, wenn es zunächst die XML-Deklaration zu Beginn des Dokumentes enthält, wie in Aufgabe 2 zu sehen sein wird.

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

Die XML-Deklaration beinhaltet die Versionsnummer, welches sich auf die Sprachspezifikation von XML bezieht. Derzeit ist die Version 1.0 von XML maßgeblich. Zudem kann das Dokument auch ein Attribut besitzen, welches zur Verwendung von Zeichenketten "encoding" hinzugefügt wird.

Jedes wohlgeformte XML-Dokument muss mindestens ein Datenelement haben:

```
<?xml version="1.0" encoding="UTF-8"?>
<anmeldung xmlns="www.bliblablup/anmeldung.de">
  <Schlagzeugwettbewerb>
</Schlagzeugwettbewerb>
```

ODER

```
  <xs:schema attributeFormDefault="unqualified"
    elementFormDefault="qualified"
targetNamespace="http://www.chefkoch.de/rezepte/24641006006067/Lenchen-s-Schokoladenkuchen.html" xmlns:xs="http://www.w3.org/2001/XMLSchema">
    <xs:element name="Recipe">
      <xs:complexType>
        <xs:sequence>
</xs:sequence>
      </xs:complexType>
</xs:element>
</xs:schema>
```

Validität

Beispiel:

```
<?xml version="1.0"?> <Dialog>
<Adam Emotion="heftig">ich liebe dich!</Adam>
<Eva Emotion="heftig">ich dich auch!</Eva> </Dialog>
```

Valide XML-Instanzen müssen wohlgeformt sein, und darüber hinaus von einer DTD oder einem XML-Schema abgeleitet sein und allen Deklarationen des Inhaltsmodells, die in der DTD oder dem XML-Schema definiert wurden, entsprechen.

2.2

Aufgabe 2

a) 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.

Vorgang/Erläuterung

Um einen angemessenen einstieg in XML zu finden habe ich mich zunächst mit den gegebenen Tutorials befasst, die mir bei meiner Einarbeitung wesentlich geholfen haben. Basierend auf die Tutorials habe ich daraufhin mein XML-Dokument erstellt.

```
1<?xml version="1.0" encoding="UTF-8"?>
2
3
4<anmeldung xmlns="www.bliblablup/anmeldung.de">
5
6<Gruppenleiter>
7  <gruppe>
8    <gruppenleiter Vorname="" Nachname="" E-Mail=""
9Geburtsdatum="" Erfahrung="" Schlagzeug="" />
10  </gruppenleiter>
11  <Anmerkung></Anmerkung>
12  <mitglieder>
13    <mitglieder Vorname="Dietrich" Nachname="Meier"
14Email="dietrichmeiner23432345@hotmail.fr" Geburtstag="14.05.1978"
15Erfahrung="Profi" Schlagzeug="vorhanden" />
16  </mitglieder>
17 </gruppe>
18 <Anmerkung>Durchziehn! ..</Anmerkung>
19</Gruppenleiter>
20
21</anmeldung>
```

Überarbeitetes XML-Dokument

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

<Schlagzeugwettbewerb>
  <gruppe>
    <Person>
      <Vorname></Vorname>
      <Nachname></Nachname>
      <E-Mail></E-Mail>
      <Geburtsdatum></Geburtsdatum>
      <Erfahrung></Erfahrung>
      <Schlagzeug></Schlagzeug>
    </Person>
    <Anmerkung></Anmerkung>
  </gruppe>
</Schlagzeugwettbewerb>

<Schlagzeugwettbewerb>
  <gruppe>
    <Person>
      <Vorname>Dietrich</Vorname>
      <Nachname>Meier</Nachname>
      <E-Mail>dietrichmeier234324234@hotmail.fr</E-Mail>
      <Geburtsdatum>14.05.1978</Geburtsdatum>
      <Erfahrung>2</Erfahrung>
      <Schlagzeug>1</Schlagzeug>
    </Person>
    <Person>
      <Vorname>Oli</Vorname>
      <Nachname>Schmitz</Nachname>
      <E-Mail>olischmitz234535345345@hotmail.fr</E-Mail>
      <Geburtsdatum>24.04.1973</Geburtsdatum>
      <Erfahrung>3</Erfahrung>
      <Schlagzeug>2</Schlagzeug>
    </Person>
    <Anmerkung>Durchzieh! ..</Anmerkung>
  </gruppe>
</Schlagzeugwettbewerb>
```

Vorgang/Erläuterung:

Um den gegebenen Aufgabenstellungen gerecht zu werden habe ich einige Änderungen des Abgebildeten Programmcodes vorgenommen, da hier einige Fehler enthalten sind. (Vergleich mich verbessertem Code aus GitHub). Zunächst wurde in Zeile 6 der Tag <Gruppenleiter> angelegt, da ansonsten eine eindeutige Namespace nicht festgelegt werden konnte.

Um eine Kollabrierung von Vokabularen innerhalb eines Dokumentes zu vermeiden, musste in Zeile 8 das Wort „gruppenleiter“ durch „Schlagzeugwettbewerb“ ersetzt werden (Namenskonflikt).

Um die Lesbarkeit zu verbessern wurden in Zeile 8 die Elemente unter dem Tag <gruppenleiter> einzeln aufgelistet. Dies wurde auch in Zeile 13-15 vorgenommen. Unter anderem sollte über das Formular die Möglichkeit bestehen, mehrere Personen gleichzeitig zu erfassen. Um dies umzusetzen wird ein neues Tag <Person> erzeugt, welches auch in <Vorname>,

<Nachname>, <E-Mail>, <Geburtsdatum>, <Erfahrung>, <Schlagzeug>, <Anmerkung> unterteilt wurde. Zum Abschluss wurde dann das Dokument mit Datensätzen gefüllt.

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

Vorgang/Erläuterung:

Nachdem das XML-Dokument vollständig erfasst wurde, war die Überführung zu JSON relativ einfach. Natürlich wurde das JSON-Dokument nach der Änderung des XML-Dokuments angepasst.

```
1{
2"anmeldung": {
3"-xmlns": "www.bliblablup/anmeldung.de",
4"Schlagzeugwettbewerb": {
5"gruppe": {
6"gruppenleiter": {
7"Person": {
8
9},
10"mitglieder": {
11"Vorname": "Dietrich",
12"Nachname": "Meier",
13"E-Mail": "dietrichmeiner23432345@hotmail.fr",
14"Geburtsdatum": "14.05.1978",
15"Erfahrung": "Profi",
16"Schlagzeug": "vorhanden"
17},
18"Anmerkung": "Durchziehn! .."
19}
20}
21}
22}
23}
```

Arrays werden durch eckige klammern deklariert „[...]“. In JSON bilden der Elementname und dessen Wert ein Paar, welches durch den „:“ getrennt ist. Die einzelnen Objekte liegen in geschweiften Klammern „{,“ in denen mehrere Name-Werte-Paare liegen können. Die Elemente vom XML-Dokument wurden als Objekte in JSON erfasst. In Zeile 10 wird ein Array mit dem Objekt „mitglieder“ erzeugt, welches weitere 6 Objekte beinhaltet und mit einem Datensatz vervollständigt wird.

Überarbeitetes JSON-Dokument

```
{
  "Schlagzeugwettbewerb": [
    {
      "gruppe": [
        {
          "Person": [
            {
              "Vorname": "Dietrich",
              "Nachname": "Meier",
              "E-Mail": "dietrichmeiner23432345@hotmail.fr",
              "Geburtsdatum": "14.05.1978",
              "Erfahrung": 2,
              "Schlagzeug": 1
            },
            {
              "Vorname": "Oli",
              "Nachname": "Schmitz",
              "E-Mail": "olischmitz234535345345@hotmail.fr",
              "Geburtsdatum": "24.04.1973",
              "Erfahrung": 3,
              "Schlagzeug": 2
            }
          ],
          "Anmerkung": "Durchziehn! .."
        }
      ]
    }
  ]
}
```

2.3

Aufgabe 3

a) Gegeben ist folgendes Rezept:

<http://www.chefkoch.de/rezepte/24641006006067/Lenchen-s-Schokoladenkuchen.html>

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.

Vorgang

Um Rezepte zu Identifizieren sollte man zunächst jedem Rezept eine eindeutige ID zuordnen. In diesem Fall beschränkt sich die ID auf ein einziges Rezept, wie im Code zu sehen unter `<Rezept id="1">`.

Das Root Element Rezept beinhaltet alle Kindelemente wie Titel, Bild, Zutaten, Titel2, Arbeitszeit, Schwierigkeitsgrad, Brennwert, Zubereitung sowie Kommentare.

Die Zutaten wurden als Attribute definiert, da sie jeweils einzelne Werte und Einheiten besitzen. Unter `<kommentar>` werden die werte des jeweiligen Verfassers mit eingebunden wie z. B. `<name>`, `<datum>`, `<uhrzeit>` und `<text>`.

```
<Zutat Menge="200" Einheit="g" >Butter</Zutat>
<Zutat Menge="200" Einheit="g" >Zucker</Zutat>
<Zutat Menge="200" Einheit="g" >Schokolade, Blockschokolade</Zutat>
<Zutat Menge="120" Einheit="g" >Mehl</Zutat>
<Zutat Menge="1/2" Einheit="TL" >Backpulver</Zutat>
<Zutat Menge="1" Einheit="Pkt.">Vanillezucker</Zutat>
<Zutat Menge="4" Einheit="" >Ei(er)</Zutat>
```

Das selbe gilt für die Attribute „Arbeitszeit“, „Schwierigkeitsgrad“ und „Brennwert“.

```
<Arbeitszeit name="Arbeitszeit:" value="1" Einheit="Std." />
<Schwierigkeitsgrad name="Schwierigkeitsgrad:" Value="normal" />
<Brennwert name="Brennwert p.P.:" value="295" Einheit="kcal"/>
```

Die Kommentare sollten wie bei „Rezept“ mit einer ID eindeutig identifizierbar sein.

```
<Kommentar id="1"> <name>swieselchen</name> <id>1</id>
<Bild>http://static.chefkochcdn.
de/ck.de/avatar/5ab77fec95e69919037a2ea283c4d6a6- 60fix.jpg</Bild>
<Datum>07.02.2002 18:49 Uhr</Datum>
<Kommentar>Habe Deinen Kuchen gestern gebacken (kleine Abwandlung: statt
Blockschokolade hatte ich nur Kuvertüre, zartbitter und ich habe noch
etwas Rumaroma und eine Prise Salz dazugegeben) mein LAG war total
begeistert. Ich übrigens auch, super Rezept.</Kommentar>
```

b) 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.

Vorgang/Erläuterung:

Eine wichtige Gemeinsamkeit ist, dass die Auflistung der Zutaten sich nach Art und Darstellung ähneln. Das bedeutet, dass jede einzelne Angabe mit den jeweiligen Einheiten aufgelistet werden und das direkt zu Beginn nach der Abbildung des Rezeptes. Die Struktur der Seite ändert sich nicht.

Die Grundstruktur der XML-Datei bleibt gleich. Die Unterschiede liegen in den Werten, welche man erweitern bzw. durch andere Werte ersetzen kann.

Die Angabe der Portionen, die Beschreibung zur Zubereitung, Abbildungen, Kommentare von einzelnen Usern, Tags für Soziale Netzwerke ändern sich strukturell nicht.

Einige Rezepte enthalten Bilder, andere wiederum nicht diese können dennoch durch das XML- Dokument mit eingebunden werden. Schlagwörter wie Schokolade sind in einigen Rezepten verlinkt, so dass diese den User auf andere Seiten weiterleitet.

Nicht alle Daten sind in allen Rezepten angegeben. Das beste Beispiel hierfür ist die Angabe zum Brennwert, welches auch "keine Angabe" enthalten kann.

c) 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 complex-types abgebildet werden? Für welche Daten ist die Abbildung in Attributen sinnvoller? Welche Datentypen müssen für die Elemente definiert werden? Welche Restriktionen müssen definiert werden?

Vorgang/Erläuterung:

Jedem Element und Attribut werden Typen zugeordnet. Es gibt einfache Typen (simple-Type) und komplexe (complex-Type) Typen.

Der simpleType definiert einen einfachen Typ, welcher bereits vordefinierte Typen, wie z.B. xsd:string, xsd:int etc. enthält.

Komplexe Datentypen erlauben in XML die Kindelemente eines Elementes zu beschreiben.

Komplexe Typen enthalten entweder einen Inhalt (simpleContent oder complexContent) oder eines der vier Gruppierungselemente (group, all, choice, sequence) zusammen mit evtl. einem oder mehreren Attributdefinitionen (attribute oder attributeGroup).

Wenn in der Datenstruktur zur Beschreibung eines Tags nichts weiter als eine Eigenschaft berücksichtigt wird, dann reicht es, ein Element `<....>...</....>` zu definieren.

Wenn aber noch weitere Eigenschaften berücksichtigt werden sollen, dann bietet es sich an, die Datenstruktur so zu definieren, dass es ein Element mit verschiedenen Attributen gibt.

Ein Attribut kann in der Regel umgewandelt werden zu einem strukturiertem Inhalt. Denn jedes Attribut ist als Kindelement darstellbar.

Bsp.: <code><Familie name="Müller"></code>		<code><Familie></code>
<code><Vater>Paul</Vater></code>	→	<code><name> Müller </name></code>
<code></Familie></code>		<code></name></code>

Vorgang/Erläuterung:

vollständige Datei in Github.

Im Code wird deutlich das einige Kindelemente aufkommen wie z. B. Das Kindelement „Titel“ von „Rezept“. Diese Daten müssen in complexType abgebildet werden, da nur xs:complexType Kindelemente erlaubt.

Der Indikator Occurrence wird dann benutzt, um zu definieren wie oft ein Element erscheinen soll.

Wenn die Occurrence-Eigenschaft auf Restricted festgelegt ist und die ersten Zeitelemente im XML-Dokument gefunden werden, wird die Schemadeklaration als minOccurs="1" abgeleitet. Wenn Attribute gefunden werden, wird die Schemadeklaration als use="required" abgeleitet.

Wenn die Occurrence-Eigenschaft auf Relaxed festgelegt ist, werden Elementdeklarationen als minOccurs="0" abgeleitet, und Attributdeklarationen werden als use="optional" abgeleitet. Der Standardwert der Occurrence-Eigenschaft ist Restricted.

```
<xs:element name="Zutat" minOccurs="1" maxOccurs="unbounded">
<xs:complexType> <xs:simpleContent>

<xs:restriction base="xs:string">

<xs:attribute type="xs:string" name="Name"/>

<xs:attribute type="xs:int" name="Menge"/>
<xs:attribute type="xs:int" name="Einheit"/>
</xs:restriction>
</xs:simpleContent>
</xs:complexType>
</xs:element>
```

Im Code wird deutlich, dass das Element „Zutat“ mindestens einmal erscheinen soll. Mit `<xs:element name="Zutat" maxOccurs="unbounded" minOccurs="1">` legt man fest, dass unendlich viele Zutaten mit eingebunden werden können und mindestens eine Zutat verwendet werden soll. Das Element „Zutat“ enthält weitere Inhalte wie Name, Menge und Einheit, welche als Attribute geltend gemacht werden, indem man `xs:simpleContent` verwendet. Dabei muss ein Restriction definiert werden.

Genauso kann man auch ein `<extension>` -element anstatt eines `<restriction>` -elements unter dem `<simpleContent>` -element verwenden.

Die Attribute erhalten neben dem Namen auch ihre Datentypen. Zutat verwendet den Datentyp String, Menge und Einheit verwenden wiederum einen Integer-Wert.

d)

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.

Vorgang/Erläuterung: Datei in Github

XML-Schema anhand Aufgabe 3c Das XML-Dokument enthält beinahe den selben Umriss wie das XMLDokument in Aufgabe 3a. Es sollte zusätzlich mit 2 validen Datensätzen gefüllt werden. Hier habe ich ein Rezept mit den Datensätzen ID „1“ und „2435“ eingefügt.

Überarbeitetes XML-Dokument:

```
<?xml version="1.0" encoding="UTF-8" standalone="yes"?>
<Rezepte>
  <rezept id="1">
    <titel>Lenchens Schokoladenkuchen</titel>
    <bilder>
      <bild>
        <benutzer>naschkatze65</benutzer>
        <url>img/naschkatze.png</url>
      </bild>
      <bild>
        <benutzer>garfield84</benutzer>
        <url>img/garfield.png</url>
      </bild>
    </bilder>
    <zutaten portion="1">
      <zutat>
        <betrag>12.5</betrag>
        <einheit>g</einheit>
        <name>Butter</name>
      </zutat>
      <zutat>
        <betrag>12.5</betrag>
        <einheit>g</einheit>
        <name>Zucker</name>
      </zutat>
      <zutat>
        <betrag>12.5</betrag>
        <einheit>g</einheit>
        <name>Schokolade, Blockschokolade</name>
      </zutat>
    </zutaten>
  </rezept>

```

```

        <betrag>7.5</betrag>
        <einheit>g</einheit>
        <name>Mehl</name>
    </zutat>
    <zutat>
        <betrag>0.03</betrag>
        <einheit>TL</einheit>
        <name>Backpulver</name>
    </zutat>
    <zutat>
        <betrag>0.06</betrag>
        <einheit>Pck</einheit>
        <name>Vanillezucker</name>
    </zutat>
    <zutat>
        <betrag>0.25</betrag>
        <einheit>Stck</einheit>
        <name>Eier</name>
    </zutat>
</zutaten>
<vorbereitung>
    <arbeitszeit>60.0</arbeitszeit>
    <brennwert>295.0</brennwert>
    <zubereitung>Butter und Schokolade im Wasserbad schmelzen.

```

Eier

trennen. Eiweiß steif schlagen. Eigelbe, Zucker und Vanillezucker verrühren. Geschmolzene Butter-Schokomasse hinzufügen und mischen. Mehl mit dem Backpulver in die Masse sieben und zum Schluss die steifen Eiweiße vorsichtig unterheben. In eine gut gefettete Form geben.

Bei 180°Grad 40 – 50 Minuten backen.

```

        </zubereitung>
    </vorbereitung>
    <kommentare>
    </kommentare>
</rezept>

```

<Rezepte>

```

<rezept id="2">
    <titel>Lenchens Schokoladenkuchen</titel>
    <bilder>
        <bild>
            <benutzer>naschkatze65</benutzer>
            <url>img/naschkatze.png</url>
        </bild>
        <bild>
            <benutzer>garfield84</benutzer>
            <url>img/garfield.png</url>
        </bild>
    </bilder>

```

```

</bilder>
<zutaten portion="1">
  <zutat>
    <betrag>12.5</betrag>
    <einheit>g</einheit>
    <name>Butter</name>
  </zutat>
  <zutat>
    <betrag>12.5</betrag>
    <einheit>g</einheit>
    <name>Zucker</name>
  </zutat>
  <zutat>
    <betrag>12.5</betrag>
    <einheit>g</einheit>
    <name>Schokolade,Blockschokolade</name>
  </zutat>
  <zutat>
    <betrag>7.5</betrag>
    <einheit>g</einheit>
    <name>Mehl</name>
  </zutat>
  <zutat>
    <betrag>0.03</betrag>
    <einheit>TL</einheit>
    <name>Backpulver</name>
  </zutat>
  <zutat>
    <betrag>0.06</betrag>
    <einheit>Pck</einheit>
    <name>Vanillezucker</name>
  </zutat>
  <zutat>
    <betrag>0.25</betrag>
    <einheit>Stck</einheit>
    <name>Eier</name>
  </zutat>
</zutaten>
<vorbereitung>
  <arbeitszeit>60.0</arbeitszeit>
  <brennwert>295.0</brennwert>
  <zubereitung>Butter und Schokolade im Wasserbad schmelzen.

```

Eier

trennen. Eiweiß steif schlagen. Eigelbe, Zucker und Vanillezucker verrühren. Geschmolzene Butter-Schokomasse hinzufügen und mischen. Mehl mit dem Backpulver in die Masse sieben und zum Schluss die steifen Eiweiße vorsichtig unterheben. In eine gut gefettete Form geben.

Bei 180°Grad 40 – 50 Minuten backen.

```

        </zubereitung>
    </vorbereitung>
    <kommentare>
        <kommentar>
            <benutzer>
                <benutzername>swieselchen</benutzername>
                <avatar>img/swieselchen.png</avatar>
            </benutzer>
            <datum>07.02.2002 18:49 Uhr</datum>
            <nachricht>Habe Deinen Kuchen gestern gebacken (kleine

```

Abwandlung:

statt Blockschokolade hatte ich nur Kuvertüre, zartbitter und ich habe noch etwas Rumaroma und eine Prise Salz dazugegeben) mein LAG war total begeistert. Ich übrighends auch, super Rezept.

```

        </nachricht>
    </kommentar>
    <kommentar>
        <benutzer>
            <benutzername>Mane</benutzername>
            <avatar>img/Mane.png</avatar>
        </benutzer>
        <datum>07.01.2003 15:06 Uhr</datum>
        <nachricht>Habe ihn eben gebacken und bin

```

begeistert.Super Rezept:-)

LG Man

```

        </nachricht>
    </kommentar>
</kommentare>
</rezept>
</Rezepte>
</Rezepte>

```

Überarbeitete XML-Schema-Datei:

```
<xs:schema xmlns:xs="http://www.w3.org/2001/XMLSchema">
  <xs:element name="Rezepte">
    <xs:complexType>
      <xs:sequence>
        <xs:element name="rezept" maxOccurs="unbounded" minOccurs="0">
          <xs:complexType>
            <xs:sequence>
              <xs:element type="xs:string" name="titel"/>
              <xs:element type="xs:string" name="untertitel"
minOccurs="0"/>
              <xs:element name="bilder">
                <xs:complexType>
                  <xs:sequence>
                    <xs:element name="bild" maxOccurs="unbounded"
minOccurs="0">
                      <xs:complexType>
                        <xs:sequence>
                          <xs:element type="xs:string" name="benutzer"/>
                          <xs:element type="xs:string" name="url"/>
                        </xs:sequence>
                      </xs:complexType>
                    </xs:element>
                  </xs:sequence>
                </xs:complexType>
              </xs:element>
            </xs:sequence>
          </xs:complexType>
        <xs:element name="zutaten">
          <xs:complexType>
            <xs:sequence>
              <xs:element name="zutat" maxOccurs="unbounded">
                <xs:complexType>
                  <xs:sequence>
                    <xs:element name="betrag">
                      <xs:simpleType>
                        <xs:restriction base="xs:double">
                          <xs:minInclusive
value="0"/>
                        </xs:restriction>
                      </xs:simpleType>
                    </xs:element>
                    <xs:element type="xs:string" name="einheit"/>
                    <xs:element type="xs:string" name="name"/>
                  </xs:sequence>
                </xs:complexType>
              </xs:element>
            </xs:sequence>
          </xs:complexType>
        </xs:element>
      </xs:sequence>
    </xs:complexType>
  </xs:element>
</xs:schema>
```

```

        <xs:attribute name="portion">
            <xs:simpleType>
                <xs:restriction base="xs:integer">
                    <xs:minInclusive value="0"/>
                    <xs:maxInclusive value="100"/>
                </xs:restriction>
            </xs:simpleType>
        </xs:attribute>
    </xs:complexType>
</xs:element>
<xs:element name="vorbereitung">
    <xs:complexType>
        <xs:sequence>
            <xs:element type="xs:double" name="arbeitszeit"/>
            <xs:element type="xs:string"
name="schwierigkeitsgrad"/>
            <xs:element type="xs:double" name="brennwert"/>
            <xs:element type="xs:string" name="zubereitung"/>
        </xs:sequence>
    </xs:complexType>
</xs:element>
<xs:element name="kommentare">
    <xs:complexType>
        <xs:sequence>
            <xs:element name="kommentar" maxOccurs="unbounded"
minOccurs="0">
                <xs:complexType>
                    <xs:sequence>
                        <xs:element name="benutzer">
                            <xs:complexType>
                                <xs:sequence>
                                    <xs:element type="xs:string"
name="benutzername"/>
                                    <xs:element type="xs:string"
name="avatar"/>
                                </xs:sequence>
                            </xs:complexType>
                        </xs:element>
                        <xs:element type="xs:string" name="datum"/>
                        <xs:element type="xs:string" name="nachricht"/>
                    </xs:sequence>
                </xs:complexType>
            </xs:element>
        </xs:sequence>
    </xs:complexType>
</xs:element>
</xs:sequence>
</xs:complexType>
</xs:element>
<xs:attribute type="xs:int" name="id"/>
</xs:complexType>

```

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

2.4

Aufgabe 4

In dieser Aufgabe entwickeln Sie mit Hilfe des JAXB Frameworks ein Java-Programm, welches die XML-Datei aus der vorigen Aufgabe einlesen, modifizieren und ausgeben kann.

a) 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 welches hier herunter geladen werden kann: <http://sourceforge.net/projects/jaxb-builder>.

Dieses kann wie ein normales Plugin in Eclipse eingebunden werden. Zur Nutzung des Plugins klicken Sie mit der rechten Maustaste auf die Schema-Datei und wählen Sie aus dem Kontextmenü Generate =>JAXB-Classes... und folgen Sie den weiteren Anweisungen in dem Dialogfenster.

Vorgang/Erläuterung:

Nachdem das XML-Schema aus Aufgabe 3 generiert wurde, waren 2 Klassen gegeben
ObjectFactory.java,
Rezepte.java

b) 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.

```
private static int submenu(Rezept rezept) throws JAXBException {
    BufferedReader console = new BufferedReader(new InputStreamReader(
        System.in));
    try {
        System.out.println("Wollen Sie das Rezept \" " + rezept.getTitel()
            + "\" ausgeben? (0)");
        System.out.println("Wollen Sie einen Kommentar zu dem Rezept \" "
            + rezept.getTitel() + "\" abgeben? (1)");
        System.out.println("Wollen Sie das Programm abbrechen? (2)");
        int code = Integer.parseInt(console.readLine());
        if ((code >= 0) && (code < 3)) {
            return code;
        }
    } catch (IOException e) {
        // Sollte eigentlich nie passieren
        e.printStackTrace();
    }
    return 2;
}
```



```

private static void printRecipe(Rezept rezept, int portion)
    throws JAXBException {
    for (int i = 0; i < 30; i++) {
        System.out.println();
    }

    System.out.println("*****");
    System.out.println("*****");
    System.out.println(rezept.getTitel() + '\n');
    if (rezept.getUntertitel() != null)
        System.out.println(rezept.getUntertitel() + '\n');
    System.out.println("*****");

    //Ausgabe der Anzeigebilder, falls vorhanden
    if (!rezept.getBilder().getBild().isEmpty()) {
        System.out.println("Userbilder:");

        for (Bild a : rezept.getBilder().getBild()) {
            System.out.print('\t' + "User: " + a.getBenutzer());
            System.out.print('\t' + "Bild: " + a.getUrl() + '\n');
        }
        System.out.println("*****");
    } else {
        System.out.println("*****");
        System.out.println("\nKeine Anzeigebilder vorhanden");
    }
}

```

Da der Programmcode zu lang ist, haben wir darauf verzichtet, den ganzen Code abzubilden. Die Klasse besteht aus acht Methoden, wobei drei Methoden zum Marshalln bzw. zum Unmarshalln sind. In der obigen Abbildung wird deutlich, dass die Aufgabe der Aufgabenstellung entsprechend gelöst wurde. Der restliche Programmcode wurde im Wiki hochgeladen.

2.5

Aufgabe 5

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.

Es ist von Vorteil Daten in einem XML oder JSON Dokument zu erfassen, da dadurch die Verarbeitung dieser Daten erleichtert wird.

Die Verwendung von XML ist ideal, wenn der Empfänger der Daten optimal darauf eingestellt ist, zum Beispiel wenn es sich um einen SOAP Webservice handelt, der XML nach HTML oder PDF konvertiert.

Handelt es sich bei der Gegenstelle um eine in JavaScript oder PHP geschriebene Webanwendung, so wird man feststellen, dass XML hier nicht die richtige Wahl für das Datenformat ist. Hier wäre die Nutzung von JSON vorteilhafter, da JSON JavaScript Objekte erstellen kann.

XML

Vorteile:

1. Durch die Trennung von Inhalt und Darstellung können Dokumente unterschiedlicher Systeme und Anwendungen weiterverarbeitet werden.
2. XML wird von vielen Software-Herstellern unterstützt.
3. Formale Prüfung des XML-Dokumentes erfolgt nicht in der Anwendung, sondern wird mit Hilfe eines XML-Schemas (XSD-Datei) durchgeführt. Bei Änderungen muss lediglich das Schema angepasst werden.
4. Tags können lesbar gestaltet werden. Dies ist von Vorteil, wenn manuelle Nachbearbeitungen von Nachrichten erfolgen müssen.

Nachteile:

1. Daten-Overhead: durch textbasiertes Format größerer Speicherbedarf (25% bis 70%) höhere Übertragungsmenge
2. Fehlende Skalierbarkeit: schlechte Performance bei Datenhaltung mögliche Abhilfe durch Kombination mit Datenbanken (XML als Input/Output)

JSON

Vorteile:

1. Die Syntax von JSON ist einfacher gestaltet und erscheint daher oft lesbarer und insbesondere leichter schreibbar.
2. Kurze Notation und einfache Handhabung in JavaScript
3. In der Regel reduziert JSON auch den Overhead im Vergleich zu XML.
4. In XML könnten viele Werte und Eigenschaften sowohl als Attribute als auch Kindknoten beschrieben werden, was zu Problemen führen kann. In JSON kann dieses Problem nicht auftreten.

Nachteile:

1. JSON ist nicht so wie SOAP (XML) als erweiterbares Format für Datenübertragungen angelegt.
2. Keine Unterstützung von Namespaces, damit schlechte Erweiterbarkeit

PHASE 2

3.1

1. Projektbezogenes XML Schema / Schemata

Die Planung und Konzeption für das Projekt nötiger valider XML Schemata soll durchgeführt werden.

Hinsichtlich der späteren Verwendung von JAXB sollte hier in jedem Fall darauf geachtet werden, dass die XML Schemata valide sind.

Um die JAXB Klassen generieren zu können ist die Erstellung einer XML-Schemata notwendig, welches die Grundlagen für unser Projekt definiert.

Um bessere Ergebnisse zu liefern wird eine XML-Datei erzeugt, dies hat den Vorteil, dass die Erstellung der XML-Schemata einfacher umsetzbar ist.

XML werden für jede Art von Datenbeschreibung, -speicherung und -austausch genutzt.

Die leichte Lesbarkeit für Menschen und Maschinen und die Portabilität sind nur zwei von vielen Vorteilen die XML bietet.

Zu Beginn wurde nur eine XML-Datei erstellt:

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

<Application>
  <StudyNews>
    <Fachhochschulen>
      <Fachhochschule>FH Köln</Fachhochschule>
      <Fachhochschule>RFH Köln</Fachhochschule>
    </Fachhochschulen>

    <Professoren>
      <Professor>Prof. Fischer</Professor>
    </Professoren>

    <Module>
      <Modul>Medieninformatik</Modul>
      <Modul>Wirtschaftsinformatik</Modul>
    </Module>
  </StudyNews>
</Application>
```

```

<!-- falsch gedacht! :)
  <Module>
    <Modulbeschreibung>bla bla</Modulbeschreibung>
    <Modulname>Datenbanken</Modulname>
    <Prof>Prof. B. Bertelsmeier</Prof>
    <Sem>Semester: 3</Sem>
    <Prüfung>Prüfungsart: schriftlich</Prüfung>
    <Voraussetzung>Voraussetzung: erfolg. Praktikum</Voraussetzung>
    <Dateien>Dateien</Dateien>
  </Modul>
  <Modul>
    <Modulbeschreibung>bla bla bla</Modulbeschreibung>
    <Modulname>Kommunikationstechnik</Modulname>
    <Prof>Prof. H. L. Stahl</Prof>
    <Sem>Semester: 3</Sem>
    <Prüfung>Prüfungsart: schriftlich</Prüfung>
    <Voraussetzung>Voraussetzung: erfolg. Praktikum</Voraussetzung>
    <Dateien>Dateien</Dateien>
  </Modul>

  <Modul>
    <Modulbeschreibung>bla bla bla bla</Modulbeschreibung>
    <Modulname>Algorithmen und Programmieren I</Modulname>
    <Prof>Prof. F. Viktor</Prof>
    <Sem>Semester: 1</Sem>
    <Prüfung>Prüfungsart: schriftlich</Prüfung>
    <Voraussetzung>Voraussetzung: erfolg. Praktikum</Voraussetzung>
    <Dateien>Dateien</Dateien>
  </Modul>
</Module> -->

<Kommentare>
  <Kommentar id="ID-1">
    <Benutzername>sinosch</Benutzername>
    <Nachricht>printf("Hello WOrld!!");</Nachricht>
  </Kommentar>
</Kommentare>

<Nachrichten>
  <Info>Heute fällt das Datenbank Praktikum aus.</Info>
  <Info>Die VL "Kommunikationstechnik" findet im Raum 3.101 statt.</Info>
</Nachrichten>
</StudyNews>
</Application>

```

Auf die Grundidee basierend wurde jedem User die Möglichkeit gegeben Modulinhalt frei hinzuzufügen.

Dies wurde in der späteren Konzeptentwicklung verworfen, um das Hinzufügen von nicht relevanten und fehlerhaften Informationen zu vermeiden.

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

<Module>
    <Modul>Kommunikationstechnik</Modul>
    <Modul>Datenbanken</Modul>
</Module>
```

An dem aktuellen Code wird deutlich, dass User neue Module hinzufügen können, jedoch keine Berechtigten Zugriff auf die Modulinhalt haben, diese Aufgabe wird lediglich vom Admin übernommen, zudem wird die Aufgabe der Überprüfung auf Richtigkeit von hinzukommenden dem Admin übertragen.

Aufgrund eines Fehlers bezüglich des ROOT-Elements `<Application>`, (worauf später tiefer eingegangen wird) wurde die erstellte XML-Datei auf die einzelnen XML-Dateien

Fachhochschule.xml,
Kommentare.xml,
Nachrichten.xml,
Module.xml und
Professoren.xml aufgesplittet.

Die wie folgt aussehen:

Fachhochschulen.xml

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

<Fachhochschulen>
    <FH>
        <Fachhochschule>FH Köln</Fachhochschule>
        <Fachhochschule>RFH Köln</Fachhochschule>
    </FH>
    <FH>
        <Fachhochschule>FH Köln</Fachhochschule>
        <Fachhochschule>RFH Köln</Fachhochschule>
    </FH>
</Fachhochschulen>
```

Eingehend auf den Code Fachhochschulen.xml, wird den Usern das hinzufügen von neuen Elementen wie einer neuen Fachhochschule/Universität gestattet.

Kommentare.xml

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

<Kommentare>
    <Kommentar id="1">
        <Benutzername>sinosch</Benutzername>
        <Nachricht>printf("Hello WOrld!!");</Nachricht>
    </Kommentar>
</Kommentare>
```

Unter Kommentare.xml, kann jeder User Nachrichten verfassen, die mit einer eindeutigen id (Identifikationsattribut) identifiziert werden zudem beinhalten diese id's einen zugeordneten Integer-Wert, um kollisionen zu vermeiden.

Um eine kollision unterstützend zu vermeiden, wird der User aufgefordert einen Benutzernamen zur erfassten Nachricht hinzuzufügen.

Module.xml

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

<Module>
    <Modul>Kommunikationstechnik</Modul>
    <Modul>Datenbanken</Modul>
</Module>
```

Unter Module.xml können auch nur Modulbezeichnungen wie „Kommunikationstechnik“ hinzugefügt werden, die Überprüfung auf richtigkeit der Modulbezeichnungen wird regelmäßig vom Admin übernommen.

Zudem gehört Module.xml zu einer der aufgeführten drei Ressourcen, auf die im laufe der Dokumentation eingegangen wird.

Nachrichten.xml

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

<Nachrichten>
    <Info>Heute fällt das Datenbank Praktikum aus.</Info>
    <Info>Die VL "Kommunikationstechnik" findet im Raum 3.101 statt.</Info>
</Nachrichten>
```

Nachrichten.xml, auch aufgeführt als Ressource, dient zum Informationsaustausch. User besitzen die möglichkeit jederzeit neue Informationen zu verfassen und aussehnstehenden zur verfügung zu stellen.

Professoren.xml

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

<Professoren>
    <Prof>
        <Name>Prof. Fischer</Name>
        <E-Mail>fischer at gm.fh-koeln.de</E-Mail>
    </Prof>

    <Prof>
        <Name>Prof. Noss</Name>
        <E-Mail>christian.noss at fh-koeln.de</E-Mail>
    </Prof>
</Professoren>
```

Professoren.xml beinhaltet die Elemente <Name> sowie <E-Mail>, hierbei wird deutlich das neue Inhalte bezüglich der Professoren angelegt werden können, die mit Name und E-Mail deklariert werden.

Professoren.xml bildet zudem unsere letzte Ressource.

XML-Schemata

XML Schema, abgekürzt XSD, ist eine Empfehlung des W3C zum Definieren von Strukturen für XML- Dokumente. Hierdurch wird die Struktur eines XML-Dokuments beschrieben.

Eine Vielzahl von Datentypen werden mittels XSD unterstützt, die für das fortlaufende Projekt von Relevanz sind.

XSD-Dateien haben üblicherweise die Endung „.xsd“.

Aufbauend auf die vorhandenen XML-Dateien, wurden zunächst folgende XML-Schemata geschrieben:

***Fachhochschule.xsd,
Kommentare.xsd,
Module.xsd,
Nachrichten.xsd,
Professoren.xsd***

Fachhochschule.xsd

```
<xs:schema attributeFormDefault="unqualified" elementFormDefault="qualified"
  xmlns:xs="http://www.w3.org/2001/XMLSchema">
  <xs:element name="Fachhochschulen">
    <xs:complexType>
      <xs:sequence>
        <xs:element type="xs:string" name="FH" maxOccurs="unbounded" minOccurs="0"/>
      </xs:sequence>
    </xs:complexType>
  </xs:element>
</xs:schema>
```

Das Element „Fachhochschule“ besitzt ein Kindelement, welches den Namen „FH“ trägt. Es wird ein „complexType“ gesetzt, um komplizierte Elemente, wie z.B. Elemente mit Kindelementen und/oder Attributen zu definieren.

„Sequence“ definiert eine geordnete Struktur mit fester Reihenfolge. „FH“ ist vom Typ String. Es sollte möglich sein mehrere Fachhochschulen hinzuzufügen, weshalb

maxOccurs und minOccurs verwendet wird.

Wenn diese Angaben fehlen, muss das Element genau einmal vorhanden sein.

minOccurs="0" erlaubt das Fehlen dieses Elements. maxOccurs="unbounded" erlaubt eine beliebige Anzahl der Elemente.

Kommentare.xsd

```
<xs:schema attributeFormDefault="unqualified" elementFormDefault="qualified"
  xmlns:xs="http://www.w3.org/2001/XMLSchema">
  <xs:element name="Kommentare">
    <xs:complexType>
      <xs:sequence>
        <xs:element name="Kommentar">
          <xs:complexType>
            <xs:sequence>
              <xs:element type="xs:string" name="Benutzername"/>
              <xs:element type="xs:string" name="Nachricht"/>
            </xs:sequence>
            <xs:attribute type="xs:byte" name="id"/>
          </xs:complexType>
        </xs:element>
      </xs:sequence>
    </xs:complexType>
  </xs:element>
</xs:schema>
```

Das Element „Kommentar“ besitzt zwei weitere Kindelemente, diese sind „Benutzername“ und „Nachricht“.

Beides sind vom Typ string. Zusätzlich hat das Element „Kommentar“ das Attribut id als Kindelement, welches vom Typ „byte“ ist.

Wie schon zu Beginn beschrieben wurde, wird die id zur eindeutigen Identifikation von Kommentaren genutzt.

Module.xsd

```
<xs:schema attributeFormDefault="unqualified" elementFormDefault="qualified"
  xmlns:xs="http://www.w3.org/2001/XMLSchema">
  <xs:element name="Module">
    <xs:complexType>
      <xs:sequence>
        <xs:element type="xs:string" name="Modul" maxOccurs="unbounded"
minOccurs="0"/>
      </xs:sequence>
    </xs:complexType>
  </xs:element>
</xs:schema>
```

Module.xsd ist strikt einfach gestaltet. Da es nur um die Modulbezeichnung geht, hat das Element „Module“ nur ein einziges Kindelement „Modul“. Es ist ebenfalls vom Typ string und da es hier auch möglich sein sollte, mehrere Module hinzuzufügen, werden erneut maxOccurs und minOccurs verwendet.

Nachrichten.xsd

```
<xs:schema attributeFormDefault="unqualified" elementFormDefault="qualified"
  xmlns:xs="http://www.w3.org/2001/XMLSchema">
  <xs:element name="Nachrichten">
    <xs:complexType>
      <xs:sequence>
        <xs:element type="xs:string" name="Info" maxOccurs="unbounded" minOccurs="0"/>
      </xs:sequence>
    </xs:complexType>
  </xs:element>
</xs:schema>
```

Das Kindelement „Info“ besitzt ein Elternelement „Nachrichten“. „Info“ ist vom Datentyp string und wie zuvor bei den vorherigen XSD's wird hier auch maxOccurs und minOccurs verwendet.

Professoren.xsd

```
<xs:schema attributeFormDefault="unqualified" elementFormDefault="qualified"
  xmlns:xs="http://www.w3.org/2001/XMLSchema">
  <xs:element name="Professoren">
    <xs:complexType>
      <xs:sequence>
        <xs:element name="Prof" maxOccurs="unbounded" minOccurs="0">
          <xs:complexType>
            <xs:sequence>
              <xs:element type="xs:string" name="Name"/>
              <xs:element type="xs:string" name="E-Mail"/>
            </xs:sequence>
          </xs:complexType>
        </xs:element>
      </xs:sequence>
    </xs:complexType>
  </xs:element>
</xs:schema>
```

Das Element „Professoren“ hat das Unterelement bzw. Kindelement „Prof“, welches wiederum weitere Kindelemente hat, und zwar „Name“ und „E-Mail“. Beides sind vom Typ string.

Es ist von großer Bedeutung, dass die XML-Schemata wohlgeformt und valide sind, da im nächsten Schritt die JAXB-Klassen generiert werden. Sind die XSD's nicht valide, sind die JAXB-Klassen nicht verwendbar.

Erzeugung der Java-Klassen mit JAXB

Der Binding-Compiler xjc (XML-to-java-compiler) wird von JAXB bereitgestellt und generiert aus einer XML-Schema-Datei Java-Klassen und eine ObjectFactory, welche JAXB zur Erstellung der zugehörigen Java-Objekte benötigt.

JAXB (Java Architecture for XML Binding) ist eine Java-API, mit der es möglich ist Java-Klassen aus einer XML-Grammatik (z.B. in Form eines XML-Schemas) heraus zu generieren und XML SchemataInstanzen dieses Schemas auf Java-Objekte abzubilden. Diesen

Vorgang nennt man XML- Datenbindung. Ebenso können aus einem bestehenden Objektmodell XML-Daten erzeugt werden.

Da ein XML Schema mehr bzw. detaillierte Informationen über die Grammatik bereitstellen kann als einfache Java-Klassen, müssen letztere mit zusätzlichen Metainformation angereichert werden, damit später ein 1:1 Mapping von Java auf XML möglich ist.

Dies geschieht in Form von sogenannten Mapping-Annotations. Diesen Prozess (bzw. den umgekehrten) nennt man „marshalling“ (bzw. „unmarshalling“) und ist eine spezielle Form der (De-)Serialisierung. JAXB kann auch als Ersatz für DOM oder SAX benutzt werden.

Der primäre Vorteil von JAXB ist, dass XML-Daten in Java manipuliert werden können, ohne sich mit den Details der De-(Serialisierung) und des Parsens auseinanderzusetzen zu müssen.

Allerdings ist das Verfahren für sehr große Datenmengen nicht geeignet, da (bei normalen Verhalten) alle Daten aus der XML Datei komplett im Hauptspeicher als Java Objekte vorliegen.

XML Binding eignet sich für datengetriebene Anwendungen, bei denen Daten im XML Format gespeichert oder ausgetauscht werden sollen, für die Interaktion zwischen Anwendungen und zum Schreiben und Lesen von XML-Dokumenten direkt aus der Anwendung.

Erwartungsgemäß erzeugte JAXB folgende Java-Klassen:

Fachhochschulen.java

Kommentare.java

Module.java

Nachrichten.java

ObjectFactory.java

Professoren.java

Durch die von JAXB generierten Java-Klassen, ist es nun möglich auf Methoden wie `getFachhochschule()`, `getModule()` etc. zuzugreifen, welche im laufenden Projekt aufgegriffen werden.

3.2

2. Ressourcen und die Semantik der HTTP-Operationen für das Projekt

Eingangs soll eine theoretische Auseinandersetzung mit HTTP-Operationen und Ressourcen im Kontext RESTful Webservices erfolgen. Das erworbene Wissen ist Grundlage um die nächsten Schritte erfolgreich absolvieren zu können.

Es sollen projektbezogene Beschreibungen der für das Projekt benötigten Ressourcen und Operationen inklusive Begründung der Entscheidungen erstellt werden.

Ressourcen

Ressourcen bilden die zentrale Abstraktion des REST-Architekturstils und stehen im Mittelpunkt des Entwurfs. REST ist daher eine ROA (Resource Oriented Architecture). Eine Repräsentation ist dabei die Darstellung einer Ressource in einem definierten Format. Die Interaktion mit Ressourcen erfolgt immer über den Austausch von Repräsentationen. Selten eindeutig geht die Präsentation einer Ressource aus der Repräsentation hervor.

Eine Ressource wird definiert als eine durch eine gemeinsame Identifikation zusammengehaltene Menge von Repräsentationen. Ressourcen im Kontext von REST sollten nicht verstanden werden als Konzepte der Persistenzschicht, sondern als übergreifende, nach außen sichtbare Anwendungskonzepte.

Operation

Die Kernidee von REST bezüglich Verben oder Operationen ist eine gleichförmige Schnittstelle. Die formale REST-Bedingung lautet, dass es eine definierte, begrenzte Menge von Operationen gibt, die für alle Ressourcen gleichermaßen gültig ist.

Dieses Prinzip steht also im Gegensatz zu anderen Methoden, wie beispielsweise der objektorientierten Programmierung, bei der beliebig viele Operationen mit beliebiger Semantik modelliert werden dürfen.

Welche Operationen dies genau sind, ist nicht festgelegt. Da RESTful Webservices jedoch über das Internet kommunizieren, ist die konkrete Ausprägung des REST-Prinzips in HTTP naheliegend. Daher werden hier die Standardverben von HTTP 1.1 aufgeführt: GET, HEAD, PUT, POST, DELETE, OPTIONS, TRACE und CONNECT.

Die HTTP-GET-Operation holt Informationen, die durch einen URI identifiziert werden, in Form einer Entity (Repräsentation) ab. GET ist als „sicher“ definiert, d.h. der Nutzer geht mit dem Aufruf einer GET-Operation keine Verpflichtungen ein.

Die HTTP-PUT-Operation dient der Aktualisierung einer bestehenden Ressource oder dem Erzeugen einer neuen (falls noch nicht vorhanden) unter dem angegebenen URI. Informationen über den Status der Ressource können im Entity-Body der HTTP-Nachricht gesendet werden. Das Format der Repräsentation wird im Content-Type-Header hinterlegt. Die Ressource muss nur „sinngemäß“ angelegt werden. Das schließt nicht zwangsläufig eine Verarbeitung aller Daten mit ein.

Die HTTP-POST-Operation dient dem Anlegen einer neuen Ressource unterhalb eines bestimmten URI. Dabei wird nicht den endgültigen URI der anzulegenden Ressource angegeben, sondern der der übergeordneten Ressource.

Diese ist für die Erstellung der Sub-Ressource verantwortlich. Der endgültige URI wird also vom Server bestimmt und dem Client über einen Location-Header bekannt gegeben. Eine erfolgreiche POST-Operation liefert den HTTP-Statuscode „201 Created“. POST ist weder „sicher“ noch idempotent, da POST im Allgemeinen eine Zustandsänderung von Ressourcen mit sich bringt.

Da POST auch für eine beliebige Verarbeitung angewendet werden kann, für die keine andere Methode passend ist, hat die Infrastruktur keine Kenntnis über die Semantik dieser Operation, d.h. niemand außer Client und Server kennt den Sinn des Aufrufs.

Zum Entfernen einer Ressource soll die HTTP-DELETE-Operation angewandt werden. Wie GET, HEAD und PUT ist auch DELETE idempotent, aber nicht sicher. Es sei noch erwähnt, dass die Semantik der DELETE-Methode lediglich besagt, dass die Ressource unter diesem URI nicht mehr verfügbar ist. Intern darf sie weiterhin existieren.

Vor Beginn der Implementierung haben wir den Einsatz der Operationen GET, PUT, POST und DELETE in Betracht gezogen. Übersichtshalber wurde eine Tabelle zu den jeweiligen Ressourcen mit Ihren zugehörigen Http-Operationen erstellt:

Ressourcen	GET	PUT	POST	DELETE
Module	✓		✓	
Nachrichten	✓		✓	✓
Professoren	✓			

Benutzer des Service müssen sich nicht mit eigenen Daten registrieren. Im System wird Ihnen die Möglichkeit geboten über die Ressource „Nachrichten“, neue Informationen hinzuzufügen oder bestehende Nachrichten abzurufen. Dies geschieht über die GET- und POST-Methode.

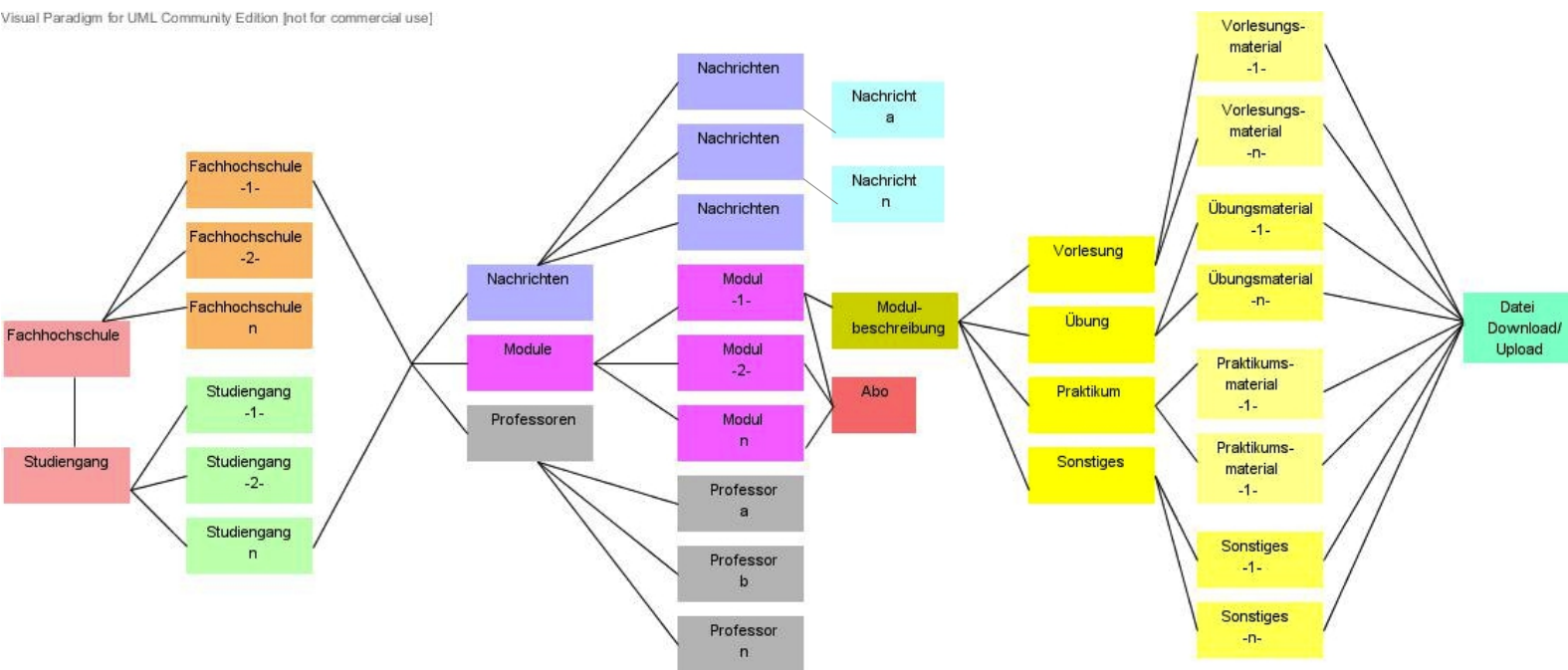
Zudem wird die Voraussetzung geschaffen, Nachrichten beliebig zu aktualisieren oder gegebenenfalls zu eliminieren, welches ebenfalls mit der GET- und POST-Methode erreicht wird.

Über die Ressource „Professoren“ können zusätzliche Auskünfte zu Professoren abgerufen werden(GET-Methode).

Die letzte Ressource „Module“ ermöglicht den Benutzern das Abrufen und das Hinzufügen von Modulbezeichnungen(GET-, POST-Methode).

Die Datenstruktur

Visual Paradigm for UML Community Edition [not for commercial use]



Anhand der Datenstruktur wurde relativ schnell ersichtlich, welche Ressourcen für das Projekt relevant sind (Nachrichten, Module, Professoren).

3.3

3. RESTful Webservice

Ein RESTful Webservice soll in Java erstellt werden. Dazu gelten folgenden Bedingungen:

Mindestens zwei Ressourcen müssen implementiert sein

JAXB soll für das marshalling / unmarshalling verwendet werden

Die Operationen GET, DELETE und POST müssen implementiert sein

Es sollen mindestens einmal PathParams und einmal QueryParams verwendet werden

RESTful Web-Service

Die Web-Services sprechen Ressourcen an und rufen entfernte Methoden auf. Ein einfaches Verfahren ist die Anfrage mit Parametern an einen Web-Server, der den Inhalt zurückliefert. Im Wesentlichen entspricht dies, was auch ein Client zum Ansprechen einer Suchmaschine macht – ein Suchstring wird dem Server geschickt und dieser erhält eine Antwort.

Beim RESTful-WebService wird dem Web-Server über HTTP eine Anfrage versendet. Die URL kodiert die Ressource, die danach eindeutig adressierbar wird. Es ist sehr wichtig, dass jede Ressource über einen eindeutigen URI (Unique Resource Identifier) identifizierbar ist, da z.B ein Kunde mit der Kundennummer 345678 über die URI <http://blupblupblup/blup/345678> adressiert werden kann. Die Ressource hat eine Repräsentation, die in jedem Format sein kann, also XML, Text, Bilder oder .mp3-Dateien.

Marshalling und Unmarshalling

Marshalling ist die Umwandlung von strukturierten oder elementaren Daten in ein bestimmtes Format. Das Unmarshalling ist die Zurückwandlung, d.h. Das Format wird auf der Empfängerseite wieder in ihre ursprüngliche Struktur wiederhergestellt. Auf dieses Projekt bezogen würde dies bedeuten, dass von JAVA zu XML gemarshallt wird und von XML zu JAVA geunmarshallt wird.

Das Marshalling und/oder Unmarshalling kann man auf verschiedenste Weise umsetzen. Entweder wird der Code zur jeweiligen Ressource separat in einer Klasse geschrieben oder es wird eine Hilfsklasse geschrieben, wo der Code von allen Ressourcen zusammen gemarshallt bzw. geunmarshallt wird:

```
package Ressourcen;

import java.io.File;
import java.io.FileNotFoundException;
import java.io.FileOutputStream;

import javax.xml.XMLConstants;
import javax.xml.bind.JAXBContext;
import javax.xml.bind.JAXBException;
import javax.xml.bind.Marshaller;
import javax.xml.bind.Unmarshaller;
import javax.xml.validation.Schema;
import javax.xml.validation.SchemaFactory;

import org.xml.sax.SAXException;

public class XMLworker {
    JAXBContext module;
    JAXBContext professoren;
    JAXBContext nachrichten;

    /**
     * Kontext fuer marshal und unmarshall
     * @throws JAXBException
     */
    public XMLworker() throws JAXBException{
        this.module = JAXBContext.newInstance(generated.Module.class);
        this.nachrichten = JAXBContext.newInstance(generated.Nachrichten.class);
        this.professoren = JAXBContext.newInstance(generated.Professoren.class);
    }
}
```

```

/**
 * Methode zum Auslesen aus der App-XML-Datei
 * @return module
 * @throws JAXBException
 */
public Module unmarshalModul() throws JAXBException{
    Unmarshaller unmarshal = module.createUnmarshaller();
    Module module = (Module) unmarshal.unmarshal(new
File("src/XSD_XML/Module.xml"));

    return module;
}

/**
 * Methode zum Schreiben in die App-XML-Datei
 * @param module
 * @throws JAXBException
 * @throws SAXException
 * @throws FileNotFoundException
 */
public void marshalModul(Module modul) throws JAXBException, SAXException,
FileNotFoundException{
    SchemaFactory schemaFactory =
SchemaFactory.newInstance(XMLConstants.W3C_XML_SCHEMA_NS_URI);
    Schema schema = schemaFactory.newSchema(new
File("src/XSD_XML/Nachrichten.xsd"));
    Marshaller marshal = module.createMarshaller();
    marshal.setSchema(schema);
    marshal.setProperty(Marshaller.JAXB_ENCODING, "UTF-8");
    marshal.setProperty(Marshaller.JAXB_FORMATTED_OUTPUT, Boolean.TRUE);
    FileOutputStream file = new
FileOutputStream("src/XSD_XML/Nachrichten.xml");
    marshal.marshal(modul, file);
}

/**

```

```

    * Methode zum Auslesen aus der App-XML-Datei
    * @return Nachrichten
    * @throws JAXBException
    */
    public Nachrichten unmarshalNachrichten() throws JAXBException{
        Unmarshaller unmarshal = nachrichten.createUnmarshaller();
        Nachrichten nachrichten = (Nachrichten) unmarshal.unmarshal(new
File("src/XSD_XML/Nachrichten.xml"));

        return nachrichten;
    }

    /**
    * Methode zum Schreiben in die App-XML-Datei
    * @param Nachrichtenliste
    * @throws JAXBException
    * @throws SAXException
    * @throws FileNotFoundException
    */
    public void marshalNachrichten(Nachrichten nachrichtenliste) throws
JAXBException, SAXException, FileNotFoundException{
        SchemaFactory schemaFactory =
SchemaFactory.newInstance(XMLConstants.W3C_XML_SCHEMA_NS_URI);
        Schema schema = schemaFactory.newSchema(new
File("src/XSD_XML/Nachrichten.xsd"));
        Marshaller marshal = nachrichten.createMarshaller();
        marshal.setSchema(schema);
        marshal.setProperty(Marshaller.JAXB_ENCODING, "UTF-8");
        marshal.setProperty(Marshaller.JAXB_FORMATTED_OUTPUT,
Boolean.TRUE);
        FileOutputStream file = new
FileOutputStream("src/XSD_XML/Nachrichten.xml");
        marshal.marshal(nachrichtenliste, file);
    }

    /**
    * Methode zum Auslesen aus der App-XML-Datei
    * @return Professoren

```

```

    * @throws JAXBException
    */
    public Professoren unmarshalProfessoren() throws JAXBException{
        Unmarshaller unmarshal = professoren.createUnmarshaller();
        Professoren professoren = (Professoren) unmarshal.unmarshal(new
File("src/XSD_XML/Professoren.xml"));

        return professoren;
    }
    /**
    * Methode zum Schreiben in die App-XML-Datei
    * @param Professoren
    * @throws JAXBException
    * @throws SAXException
    * @throws FileNotFoundException
    */
    public void marshalProfessoren(Professoren professorenliste) throws
JAXBException, SAXException, FileNotFoundException{
        SchemaFactory schemaFactory =
SchemaFactory.newInstance(XMLConstants.W3C_XML_SCHEMA_NS_URI);
        Schema schema = schemaFactory.newSchema(new
File("src/XSD_XML/Professoren.xsd"));
        Marshaller marshal = professoren.createMarshaller();
        marshal.setSchema(schema);
        marshal.setProperty(Marshaller.JAXB_ENCODING, "UTF-8");
        marshal.setProperty(Marshaller.JAXB_FORMATTED_OUTPUT,
Boolean.TRUE);
        FileOutputStream file = new
FileOutputStream("src/XSD_XML/Professoren.xml");
        marshal.marshal(professorenliste, file);
    }
}

```

Die Hilfsklasse „XMLworker.java“ enthält 6 Methoden. Drei von Ihnen sind für das Marshalling von den Ressourcen „Nachrichten“, „Module“, „Professoren“ zuständig und die anderen drei Methoden für das Unmarshalling dieser Ressourcen. Es werden drei neue Instanzen erzeugt. Zunächst wird der Programmcode geschrieben, welches zum Auslesen aus der XML-Datei notwendig ist. Darauf folgt die Methode, die zum Schreiben in die XML-Datei benötigt wird.


```

package Ressourcen;
import javax.ws.rs.*;
import javax.ws.rs.core.MediaType;
import javax.xml.bind.JAXBException;
import org.xml.sax.SAXException;

@Path("/Nachrichten")
public class Nachrichten{
    public XMLworker xmlWork;

    public Nachrichten() throws JAXBException{
        this.xmlWork = new XMLworker();
    }
    @GET
    @Produces(MediaType.APPLICATION_XML)
    public Nachrichten getNachricht() throws JAXBException, SAXException{
        return this.xmlWork.unmarshalNachrichten();
    }
    @POST
    @Path("/add")
    @Consumes(MediaType.APPLICATION_XML)
    public String postNachrichten(
        @QueryParam("nachricht") String nachricht) {
        if (nachricht == null) {
            throw new IllegalArgumentException("Nachricht darf nicht leer sein.");
        }
        return nachricht;
    }
    @DELETE
    @Path("/delete")
    public Nachrichten deleteNachrichten() {
        return null;
    }
}

```

Jede einzelne Ressource enthält Ihre eigene Klasse. Die Klasse „Nachrichten“ weist die Operationen GET , POST und DELETE vor. PathParams und QueryParams wurden jeweils in den Ressourcen eingeführt.

GET fordert die angegebene Ressource vom Openfire-Server an. Der Zustand am Server wird dabei nicht verändert, weshalb GET als sicher bezeichnet wird.

Die POST-Operation fügt eine neue Sub-Ressource unterhalb der angegebenen Ressource ein. Die übergeordnete Ressource adressiert den URI, da die neue Ressource noch kein URI besitzt. Als Ergebnis wird der neue Ressourcenlink dem Client zurückgegeben.

DELETE ist zum Löschen einer Ressource. Eine Ressource wird meist nicht physisch gelöscht, sondern nur als gelöscht gekennzeichnet und somit versteckt und deaktiviert. Eine Ressource zu löschen, die nicht existiert bzw. bereits gelöscht wurde bringt dem Client somit keine Fehlermeldung.

Module

```
package Ressourcen;
```

```
import javax.ws.rs.*;
import javax.ws.rs.core.MediaType;
import javax.xml.bind.JAXBException;
import org.xml.sax.SAXException;
```

```
@Path("/Module")
public class Module{
    public XMLworker xmlWork;

    public Module() throws JAXBException{
        this.xmlWork = new XMLworker();
    }

    @GET
    @Produces("application/xml")
    public Module getModul() throws JAXBException, SAXException{
        return this.xmlWork.unmarshalModul();
    }

    @GET
    @Path("/Module/{id}")
    @Produces("application/xml")
    public Module getModul(@PathParam("id") int S_id) throws JAXBException,
        SAXException{
```

```

        Module mod = new Module();

        Module returnModul = new Module();
        Module mdl = this.xmlWork.unmarshalModul();

        return mdl;
    }
    @POST
    @Path("/Module/add")
    @Consumes(MediaType.APPLICATION_XML)
    public String postModule(
        @QueryParam("modul") String modul) {

        if (modul == null) {
            throw new IllegalArgumentException
                ("Modulbezeichnung darf nicht leer sein.");
        }

        return modul;
    }
}

```

Professoren

```

package Ressourcen;

import javax.ws.rs.GET;
import javax.ws.rs.Path;
import javax.ws.rs.Produces;
import javax.ws.rs.core.MediaType;
import javax.xml.bind.JAXBException;

```

```
import org.xml.sax.SAXException;

@Path("/Professoren")
public class Professoren{
    public XMLworker xmlWork;

    public Professoren() throws JAXBException{
        this.xmlWork = new XMLworker();
    }

    @GET
    @Produces(MediaType.APPLICATION_XML)
    public Professoren getProfessoren() throws JAXBException, SAXException{

        return this.xmlWork.unmarshalProfessoren();
    }
}
```

3.4

4. Konzeption + XMPP Server einrichten

Es handelt sich hierbei um konzeptionellen Meilenstein für die genauere Planung der asynchrone Kommunikation. Dazu sollen folgende Aufgaben erledigt werden:

Leafs (Topics) sollen für das Projekte definiert werden

Wer ist dabei Publisher und wer Subscriber?

Welche Daten sollen übertragen werden?

Zusätzlich soll der XMPP Server installiert und konfiguriert werden.

Publish Subscribe Paradigma

Dieses Paradigma beschreibt ein System, wo Informationen zur Verfügung gestellt werden, die an interessierte Clients gesendet werden können. Diese Clients werden auch *Hörer* oder *listener* genannt. Der *Publisher* veröffentlicht die Information, der *Subscriber* bekommt sie, wenn er sie abonniert hat. Dies geschieht über *topic-based publish-subscribe* oder *content-based publish-subscribe*, wobei *topic-based* noch unterteilt wird in *listenbasiert* und *broadcast*:

Node:

Ein Knotenpunkt in einem System, über welchen Informationen bereitgestellt werden können. Alle an das System angeschlossenen Clients können auf die Nodes zugreifen und die Informationen abfragen.

Der Informationsaustausch zwischen den Clients findet über eine Node statt. Wir haben uns entschieden, eine einzige Node anzulegen, welche von allen Usern genutzt wird, um die Informationen bereit zu stellen und ab zu fragen.

Alle User sollten in der Lage sein auf die Node zu schreiben und diese abonnieren zu können. Es muss gewährleistet sein, dass das Schreiben aller User sich nicht gegenseitig behindert und dass die Informationen client-spezifisch abgerufen werden können.

Wer ist dabei Publisher und wer Subscriber?

In unserem Anwendungsbeispiel ist der Client der Publisher, denn der Client muss zu dem erfolgreichen Post diese veröffentlichen. In dem Moment ist er Publisher. Subscriber sind in dem Moment die restlichen Clients bzw. User, die sich die Informationen/Kommentare/Nachrichten vom Publisher bekommen.

Welche Daten sollen mittels der Anwendung übertragen werden?

Ein Client veröffentlicht seinen Post als XML- file. Diese enthält Informationen entweder über Neuigkeiten aus der FH oder neuen Modulen. Es könnte sich auch nur um ein Kommentar handeln.

3.5

5. XMPP - Client

Es soll ein XMPP Client entwickelt werden, dafür soll folgendes berücksichtigt werden:

Mittels der Anwendung soll es möglich sein Leafs zu abonnieren, Nachrichten zu empfangen und zu veröffentlichen.

Ermöglichen Sie die Übertragung von Nutzdaten (Beispielsweise Simplepayload)

Leafs und ggf. mögliche Eigenschaften der Leafs sollen angezeigt werden können (Service Discovery)

Publisher

```
package pubsub;

import org.jivesoftware.smack.ConnectionConfiguration;
import org.jivesoftware.smack.XMPPConnection;
import org.jivesoftware.smack.XMPPException;
import org.jivesoftware.smackx.pubsub.Item;
import org.jivesoftware.smackx.pubsub.LeafNode;
import org.jivesoftware.smackx.pubsub.PayloadItem;
import org.jivesoftware.smackx.pubsub.PubSubManager;
import org.jivesoftware.smackx.pubsub.SimplePayload;

public class Publisher {

    private static String host = "localhost";
    private static int port = 5222;

    private XMPPConnection conn;
    private PubSubManager mgr;

    public void connect(String user) throws XMPPException{
        connect(host, port, user);
    }

    /**
     * Aufbauen einer Verbindung zum Server mit entsprechender Adresse.
     * Zusätzlich wird direkt ein User mit entsprechendem Passwort eingeloggt.
     * @param host, IP-Adresse des Servers zu dem Verbunden werden soll
     * @param port, Portnummer des Servers zu der Verbunden werden soll
     * @param user, Nutzernamen zum Einloggen
     * @throws XMPPException
     */
    public void connect(String host, int port, String user) throws XMPPException{
```

```

        ConnectionConfiguration config = new ConnectionConfiguration(host, port);
        this.conn = new XMPPConnection(config);
        conn.connect();
        this.mgr = new PubSubManager(conn, "pubsub." + host);
    }

    /* Trennen der Verbindung zum Server.
    */
    public void disconnect(){
        conn.disconnect();
    }

    public void addMessage(String messageld, String nodeld) throws XMPPException {
        LeafNode node = (LeafNode) mgr.getNode(nodeld);
        node.publish(new Item(messageld));
        System.out.println("Item wurde erzeugt.");
    }

    PayloadItem<SimplePayload> item = new
    PayloadItem<SimplePayload>(messageld, payload);
    node.publish(item);
    System.out.println("Item wurde erzeugt.");
    }

    public void deleteMessage(String messageld, String nodeld) throws XMPPException{
        LeafNode node = (LeafNode) mgr.getNode(nodeld);
        node.deleteItem(messageld);
    }
}

```

Diese Klasse stellt alle wesentlichen Methoden für den Publisher bei dem PubSub Paradigma zur Verfügung. Es wird zunächst über die Methode „connect(String user)“ eine Verbindung mit dem Openfire-Server der lokal verwendet wird aufgebaut.

Die Methode „addMessage(String messageld, String nodeld)“ fügt eine neue Nachricht ohne Payload zum entsprechenden Knoten hinzu.

Im Gegensatz fügt die Methode „addPayloadMessage(String messageld, String nodeld)“ eine neue Nachricht mit XML-Payload zum entsprechenden Knoten hinzu. Zum Löschen vorhandener Nachrichten ist die Methode deleteMessage(String messageld, String nodeld) zuständig.

Subscriber

```
package pubsub;

import java.util.ArrayList;
import java.util.Iterator;
import java.util.List;

import org.jivesoftware.smack.ConnectionConfiguration;
import org.jivesoftware.smack.XMPPConnection;
import org.jivesoftware.smack.XMPPException;
import org.jivesoftware.smackx.ServiceDiscoveryManager;
import org.jivesoftware.smackx.packet.DiscoverItems;
import org.jivesoftware.smackx.pubsub.Item;
import org.jivesoftware.smackx.pubsub.LeafNode;
import org.jivesoftware.smackx.pubsub.PubSubManager;

public class Subscriber {

    private static String host = "localhost";
    private static int port = 5222;

    private XMPPConnection conn;
    private PubSubManager mgr;
    private ServiceDiscoveryManager sdMgr;

    public void connect(String user) throws XMPPException{
        connect(host, port, user);
    }

    /**
     * Aufbauen einer Verbindung zum Server mit entsprechender Adresse.
     * @param host, IP-Adresse des Servers zu dem Verbunden werden soll
     * @param port, Portnummer des Servers zu der Verbunden werden soll
     * @param user, Nutzernamen
     * @throws XMPPException
     */
    public void connect(String host, int port, String user) throws XMPPException{
        ConnectionConfiguration config = new ConnectionConfiguration(host, port);
        this.conn = new XMPPConnection(config);
        conn.connect();
        this.mgr = new PubSubManager(conn, "pubsub.localhost");
    }

    public void disconnect(){
        conn.disconnect();
    }
}
```

```

public void abonnieren(String nodeId, String user) throws XMPPException{
    LeafNode node = (LeafNode) mgr.getNode(nodeId);
    node.addItemEventListener(new ItemEventCoordinator<Item>());
    node.addItemDeleteListener(new ItemDeleteCoordinator<Item>());
    node.subscribe(user + "@" + host);
}

public List<String> getNodes() throws XMPPException{
    this.sdMgr = ServiceDiscoveryManager.getInstanceFor(conn);
    List<String> list = new ArrayList<String>();
    for (Iterator<DiscoverItems.Item> iterator = sdMgr.discoverItems("pubsub." +
        host).getItems(); iterator.hasNext();) {
        DiscoverItems.Item item = (DiscoverItems.Item) iterator.next();
        list.add(item.getNode());
    }
    for (int i = 0; i < list.size(); i++) {
        System.out.println(list.get(i));
    }
    return list;
}
}

```

Die Klasse „Subscriber“ stellt alle notwendigen Methoden für den Subscriber bei dem PubSub Paradigma zur Verfügung, um den Publisher testen und prüfen zu können. Die Methode „connect(String)“ soll die Verbindung mit dem Openfire-Server, der lokal verwendet wird aufbauen. „disconnect()“ ist zum Trennen der Verbindung. Die Methode „abonnieren(String nodeId, String user)“ ist für das Abonnieren von Nachrichten eines Knotens zuständig und die letzte Methode „getNodes()“ gibt alle vorhandenen Knoten in einer Liste zurück, die unter der bestehenden Verbindung zu finden sind.

NodeService

```
package pubsub;
import java.util.ArrayList;
import java.util.Iterator;
import java.util.List;

import org.jivesoftware.smack.ConnectionConfiguration;
import org.jivesoftware.smack.XMPPConnection;
import org.jivesoftware.smack.XMPPException;
import org.jivesoftware.smackx.ServiceDiscoveryManager;
import org.jivesoftware.smackx.packet.DiscoverItems;
import org.jivesoftware.smackx.pubsub.AccessModel;
import org.jivesoftware.smackx.pubsub.ConfigureForm;
import org.jivesoftware.smackx.pubsub.FormType;
import org.jivesoftware.smackx.pubsub.Item;
import org.jivesoftware.smackx.pubsub.LeafNode;
import org.jivesoftware.smackx.pubsub.Node;
import org.jivesoftware.smackx.pubsub.PubSubManager;
import org.jivesoftware.smackx.pubsub.PublishModel;

public class NodeService {

    private static String host = "localhost";
    private static int port = 5222;

    private XMPPConnection conn;
    private PubSubManager mgr;
    private ServiceDiscoveryManager sdMgr;

    public void connect(String user) throws XMPPException{
        connect(host, port, user);
    }

    public void connect(String host, int port, String user) throws XMPPException{
        ConnectionConfiguration config = new ConnectionConfiguration(host, port);
        this.conn = new XMPPConnection(config);
        conn.connect();
        conn.login(user, host);
        this.mgr = new PubSubManager(conn, "pubsub." + host);
    }

    /**
     * Trennen der Verbindung zum Server.
     */
    public void disconnect(){
        conn.disconnect();
    }
}
```

```
}
```

```
public List<String> getNodes() throws XMPPException{
    this.sdMgr = ServiceDiscoveryManager.getInstanceFor(conn);
    List<String> list = new ArrayList<String>();
    for (Iterator<DiscoverItems.Item> iterator = sdMgr.discoverItems("pubsub." +
        host).getItems(); iterator.hasNext();) {
        DiscoverItems.Item item = (DiscoverItems.Item) iterator.next();
        list.add(item.getNode());
    }
    for (int i = 0; i < list.size(); i++) {
        System.out.println(list.get(i));
    }
    return list;
}
```

```
public void createNode(String nodeId) throws XMPPException{
    ConfigureForm form = new ConfigureForm(FormType.submit);
    form.setAccessModel(AccessModel.open);
    form.setDeliverPayloads(false);
    form.setNotifyRetract(true);
    form.setPersistentItems(true);
    form.setPublishModel(PublishModel.open);

    mgr.createNode(nodeId, form);
}
```

```
public void createPayloadNode(String nodeId) throws XMPPException{
    ConfigureForm form = new ConfigureForm(FormType.submit);
    form.setAccessModel(AccessModel.open);
    form.setDeliverPayloads(true);
    form.setNotifyRetract(true);
    form.setPersistentItems(true);
    form.setPublishModel(PublishModel.open);

    Node node = mgr.createNode(nodeId, form);
    node.addItemDeleteListener(new ItemDeleteCoordinator<Item>());
    node.addItemEventListener(new ItemEventCoordinator<Item>());
}
```

```
public void deleteNode(String nodeId) throws XMPPException{
    mgr.deleteNode(nodeId);
}
```

```
public void configureNode(String nodeId) throws XMPPException{
    LeafNode node = (LeafNode) mgr.getNode(nodeId);
```

```

        ConfigureForm form = new ConfigureForm(FormType.submit);
        form.setAccessModel(AccessModel.open);
        form.setDeliverPayloads(true);
        form.setNotifyRetract(true);
        form.setPersistentItems(false);
        form.setPublishModel(PublishModel.open);

        node.sendConfigurationForm(form);
    }

    /**
     *
     * @param typ
     * @throws XMPPException
     */
    public void printNode(String typ) throws XMPPException{
        LeafNode node = (LeafNode) mgr.getNode(typ);
        System.out.println("Knoten-ID: " + node.getId());
        System.out.println("Item-Liste: " + node.getItems());
        System.out.println("Payload von Element 0: " + node.getItems().get(0).toXML());
        System.out.println("Knoten-Konfig: " + node.getNodeConfiguration());
    }
}

```

Die Klasse „NodeService“ dient zum Erzeugen, Verwalten, Finden und Löschen der Knotenpunkte. Wie in den anderen beiden Klassen ist die Methode „connect(String user)“ zum Aufbauen einer Verbindung mit dem Openfire-Server. „disconnect()“ ist zum Trennen der Verbindung. Die Methode „getNodes()“ holt sich alle bekannten Knotenpunkte und gibt diese in einer Liste zurück. Ebenfalls existiert in der Klasse eine Methode, die einen neuen Knoten ohne Payload erzeugt („createNode(String nodeId“).

Die Methode createPayloadNode(String nodeId) erzeugt einen neuen Knoten der Payload zurückliefert.

Um den angegebenen Knoten löschen zu können wird die Methode deleteNode(String nodeId) aufgerufen. Mithilfe der Methode configureNode(String nodeId) wird die Konfiguration der genannten Knotens geändert.

3.6

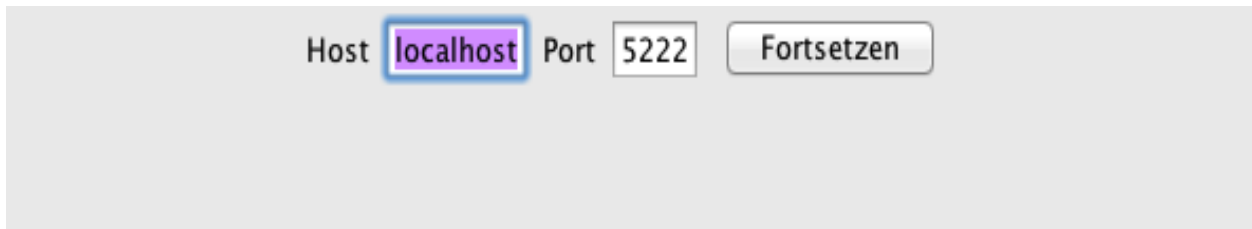
6. Client - Entwicklung

Abschließend soll ein Client erstellt werden, welcher das Projekt repräsentiert. Dafür ist eine Nutzung des REST Webservices und des XMPP Servers erforderlich:

Das System (RESTful Webservices sowie XMPP Server) muss über ein grafisches User Interface nutzbar sein. Verwendung von Swing wird empfohlen.
Getrenntes Ausführen auf unterschiedlichen Systemen von Client und Server /Webservice sollte optimalerweise möglich sein.

Das Hauptfenster Login_Main übernimmt folgende Funktionen :

- Aufnahme der Kommunikation mit dem Server mittels Hostname und Port.




The screenshot shows a section of the Login_Main window. It contains two text input fields: 'Host' with the value 'localhost' and 'Port' with the value '5222'. To the right of these fields is a button labeled 'Fortsetzen'.

Nach dem erfolgreichen Verbinden zum Server über den Host- und Portnamen, wird der Nutzer auf die Folge-Seite geleitet. Hier werden drei Auswahlmöglichkeiten gegeben: Module, News sowie Professoren.

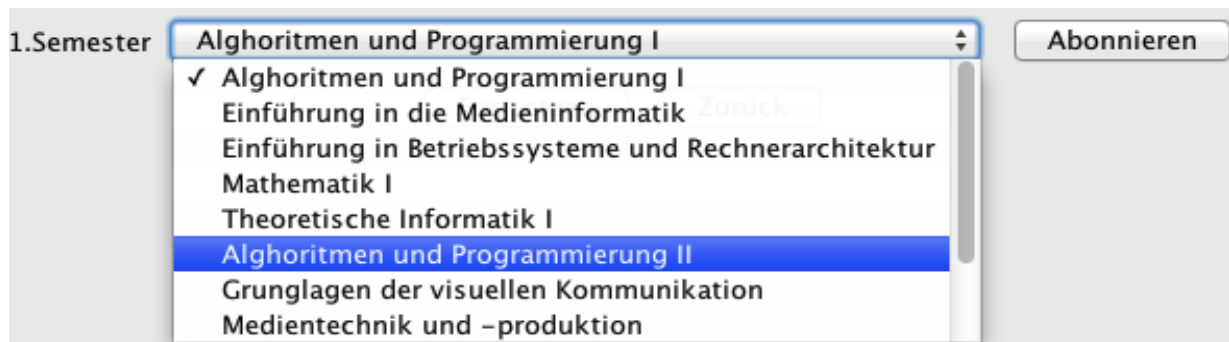


The screenshot shows a section of the application with three buttons: 'Module', 'News', and 'Professoren'. The 'Module' button is highlighted with a blue border.

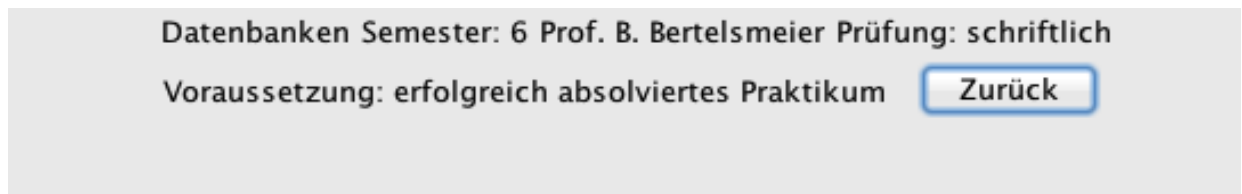
Bei Auswahl „Module“, kann der Nutzer über die Combobox, das gesuchte Modul auswählen oder Abonnieren.



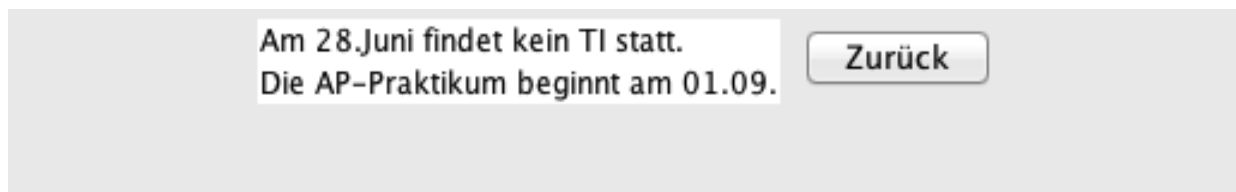
The screenshot shows a section of the application. On the left, there is a label '1.Semester'. To its right is a combobox with the text 'Algorithmen und Programmierung I'. To the right of the combobox is a button labeled 'Abonnieren'. Below the combobox are two buttons: 'Fortsetzen' and 'Zurück'.



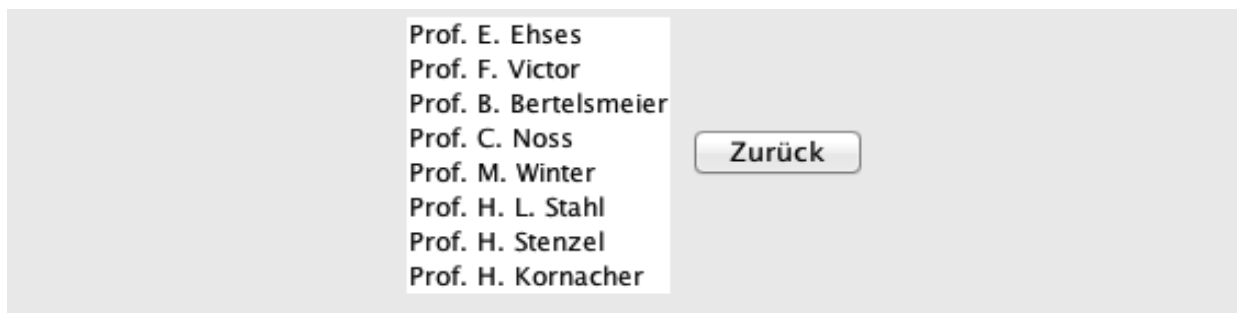
Nach Auswahl des gewünschten Moduls erscheint eine kurze Modulbeschreibung. Wie im folgenden Abbild sichtbar.



Bei Auswahl „News“, erscheint eine Liste mit den neu erstellten Nachrichten.



Nach Auswahl „Professoren“, wird eine Liste mit den Professoren und Informationen über diese ausgegeben.



4 Fazit

Unserer Meinung nach ist das Ergebnis der zweiten Phase eher unzufriedenstellend.

Aufgrund eines Fehlers mit dem Server, haben wir es nicht geschafft, unser Anwendungsbeispiel im Rahmen der Aufgabenstellungen lauffähig zu gestalten.

```
AppServer [Java Application] /System/Library/Java/JavaVirtualMachines/1.6.0.jdk/Contents/Home/bin/java (22.06.2013 21:27:24)
INFO: Starting Grizzly Framework 1.9.12 - Sun Jun 22 21:27:24 CEST 2013
URL: http://localhost:1922
22.06.2013 21:27:35 com.sun.jersey.spi.container.ContainerResponse mapMappableContainerException
SCHWERWIEGEND: The RuntimeException could not be mapped to a response, re-throwing to the HTTP container
java.lang.ClassCastException: generated.Module cannot be cast to Ressourcen.Module
    at Ressourcen.XMLworker.unmarshalModul(XMLworker.java:43)
    at Ressourcen.Module.getModul(Module.java:24)
    at sun.reflect.NativeMethodAccessorImpl.invoke0(Native Method)
    at sun.reflect.NativeMethodAccessorImpl.invoke(NativeMethodAccessorImpl.java:39)
    at sun.reflect.DelegatingMethodAccessorImpl.invoke(DelegatingMethodAccessorImpl.java:25)
    at java.lang.reflect.Method.invoke(Method.java:597)
    at com.sun.jersey.server.impl.model.method.dispatch.AbstractResourceMethodDispatchProvider$TypeOutInvoker._dispatch(AbstractResourceMethodDispatchProvider.java:124)
    at com.sun.jersey.server.impl.model.method.dispatch.ResourceJavaMethodDispatcher.dispatch(ResourceJavaMethodDispatcher.java:77)
    at com.sun.jersey.server.impl.uri.rules.HttpMethodRule.accept(HttpMethodRule.java:259)
    at com.sun.jersey.server.impl.uri.rules.ResourceClassRule.accept(ResourceClassRule.java:83)
    at com.sun.jersey.server.impl.uri.rules.RightHandPathRule.accept(RightHandPathRule.java:133)
    at com.sun.jersey.server.impl.uri.rules.RootResourceClassesRule.accept(RootResourceClassesRule.java:71)
    at com.sun.jersey.server.impl.application.WebApplicationImpl._handleRequest(WebApplicationImpl.java:990)
    at com.sun.jersey.server.impl.application.WebApplicationImpl.handleRequest(WebApplicationImpl.java:941)
    at com.sun.jersey.server.impl.application.WebApplicationImpl.handleRequest(WebApplicationImpl.java:932)
    at com.sun.jersey.server.impl.container.grizzly.GrizzlyContainer.service(GrizzlyContainer.java:132)
    at com.sun.jersey.server.impl.container.grizzly.GrizzlyAdapter.service(GrizzlyAdapter.java:166)
    at com.sun.grizzly.http.ProcessorTask.invokeAdapter(ProcessorTask.java:753)
    at com.sun.grizzly.http.ProcessorTask.doProcess(ProcessorTask.java:661)
    at com.sun.grizzly.http.ProcessorTask.process(ProcessorTask.java:914)
    at com.sun.grizzly.http.DefaultProtocolFilter.execute(DefaultProtocolFilter.java:166)
    at com.sun.grizzly.DefaultProtocolChain.executeProtocolFilter(DefaultProtocolChain.java:135)
    at com.sun.grizzly.DefaultProtocolChain.execute(DefaultProtocolChain.java:102)
    at com.sun.grizzly.DefaultProtocolChain.execute(DefaultProtocolChain.java:88)
    at com.sun.grizzly.http.HttpProtocolChain.execute(HttpProtocolChain.java:76)
    at com.sun.grizzly.ProtocolChainContextTask.doCall(ProtocolChainContextTask.java:53)
    at com.sun.grizzly.SelectionKeyContextTask.call(SelectionKeyContextTask.java:57)
    at com.sun.grizzly.ContextTask.run(ContextTask.java:69)
    at com.sun.grizzly.util.FixedThreadPool$BasicWorker.dowork(FixedThreadPool.java:379)
    at com.sun.grizzly.util.FixedThreadPool$BasicWorker.run(FixedThreadPool.java:360)
    at java.lang.Thread.run(Thread.java:680)
22.06.2013 21:27:35 com.sun.grizzly.tcp.http11.GrizzlyAdapter service
SCHWERWIEGEND: service exception
java.lang.ClassCastException: generated.Module cannot be cast to Ressourcen.Module
    at Ressourcen.XMLworker.unmarshalModul(XMLworker.java:43)
```

Und somit konnten wir auch nicht alle Aufgaben fertigstellen.

Wir sehen den Gesamterfolg des Projektes jedoch trotzdem als sehr erfolgreich an.

Wir haben als Gruppe einen sehr guten Lernerfolg erreicht. Wir haben ein Verständnis von verteilten Systemen und den unterschiedlichen Paradigmen, welche als Konzepte dahinter liegen, erlangt.

Wir betrachten unser Ergebnis der Projektarbeit in Relation zum Projektzeitraum. Für das Ziel, was wir uns gesetzt haben, hätten wir mehr Zeit benötigt. Auch das war ein Lernerfolg, da wir nun Projektarbeit und ein Zeit-Aufwand-Verhältnis besser einzuschätzen gelernt haben.

Wir hoffen, dass trotz der Nichtfertigstellung der Aufgaben unser Lernprozess deutlich geworden ist und sich das in unserer Gesamtnote wieder finden lässt.

Quellenangabe:

<http://www.alexander-stelter.de/blog/9-javascript-object-notation-json/>

http://www.nds.rub.de/media/hfs/attachments/files/2010/03/hackpra09_heid_webappsec3.pdf

http://de.wikipedia.org/wiki/Extensible_Markup_Language Quelle: <http://de.softuses.com/73650>

[http://de.wikipedia.org/wiki/Namensraum_\(XML\)](http://de.wikipedia.org/wiki/Namensraum_(XML))

http://www.come2xml.de/was_ist_wohlgeformt.html

<http://de.selfhtml.org/xml/regeln/begriffe.html>

<http://www.gm.fh-koeln.de/~vsch/anmeldung/gruppenanmeldung.html>

<http://www.w3schools.com/xml/>

<http://code-inside.de/blog/artikel/guide-xml-xml-schema-xsd-teil-2/>

[http://msdn.microsoft.com/dede/library/system.xml.schema.xmlschemainference.occurrence\(v=vs.80\).aspx](http://msdn.microsoft.com/dede/library/system.xml.schema.xmlschemainference.occurrence(v=vs.80).aspx)

<http://www.youtube.com/watch?v=5Jt7kihFmA&list=FL4xUWm5FnMYXUFkYxMjGsCQ&index=>

http://de.wikipedia.org/wiki/Java_Architecture_for_XML_Binding#

<http://www.vogella.com/articles/JAXB/article.html>

<http://www.torsten-horn.de/techdocs/java-xml-jaxb.html>

http://de.wikipedia.org/wiki/Microsoft_Publisher

<http://www.java-tutorial.org/gui-programmierung.html>