HTWK Leipzig

Fakultät für Informatik, Mathematik & Naturwissenschaften

Bachelorarbeit

Generierung und Design einer Client-Bibliothek für einen RESTful Web Service am Beispiel der Spreadshirt-API

Andreas Linz

10INB-T

admin@klingt.net

Leipzig, 28. Oktober 2013

Gutachter:

Dr. rer. nat. Johannes Waldmann

HTWK Leipzig – Fakultät für Informatik, Mathematik & Naturwissenschaften

waldmann@imn.htwk-leipzig.de

HTWK Leipzig, F-IMN, Postfach 301166, 04251 Leipzig

Jens Hadlich

Spreadshirt HQ, Gießerstraße 27, 04229 Leipzig

jns@spreadshirt.net

Diese Seite wurde mit Absicht leer gelassen.

Andreas Linz
Nibelungenring 52
04279 Leipzig
admin@klingt.net
www.klingt.net

Generierung und Design einer Client-Bibliothek für einen RESTful Web Service am Beispiel der Spreadshirt-API
Bachelorarbeit, HTWK Leipzig, 28. Oktober 2013

made with XeTeX and BibTeX.

Selbständigkeitserklärung

Ich erkläre hiermit, dass ich diese Bachelorarbeit selbstständig ohne Hilfe Dritter und ohne Benutzung anderer als der angegebenen Quellen und Hilfsmittel verfasst habe. Alle den benutzten Quellen wörtlich oder sinngemäß entnommenen Stellen sind als solche einzeln kenntlich gemacht.
Diese Arbeit ist bislang keiner anderen Prüfungsbehörde vorgelegt und auch nicht veröffentlicht worden.
Ich bin mir bewusst, dass eine falsche Erklärung rechtliche Folgen haben wird.

.....
Andreas Linz
Leipzig, 28. Oktober 2013

Danksagungen

Ein besonderer Dank gilt Dr. rer. nat. Johannes Waldmann für die Anregungen und Ratschläge sowie das gezeigte Interesse am Thema.

Ebenso möchte ich mich bei Jens Hadlich bedanken, der für Fragen jederzeit ein offenes Ohr hatte und mir half den »roten Faden« bei der Strukturierung der Arbeit nicht zu verlieren. Außerdem gilt Spreadshirt ein großer Dank, da sie mir eine wunderbare Arbeitsumgebung sowie nette und hilfreiche Kollegen zur Verfügung gestellt haben.

Für das schnelle und aufmerksame Korrekturlesen möchte ich mich an dieser Stelle noch bei Elisa Jentsch bedanken.

Abstract

German

Die vorliegende Arbeit beschreibt den Entwurf eines Codegenerators mit austauschbarer Zielsprache, der aus der abstrakten Beschreibung der Spreadshirt-API eine Client-Bibliothek generieren soll. Die Implementierung des Codegenerators erfolgt in Java, die Zielsprache der Bibliothekt ist PHP.

Es werden die Grundlagen von WebServices, Codegeneratoren und Dokumentbeschreibungssprachen erläutert und darauf aufbauend Datenmodelle erstellt welche die Beschreibung der Spreadshirt-API für den Generator enthalten. Außerdem wird das erstellte Sprachenmodell betrachtet, welches die Konstrukte der zu erzeugenden Zielsprache kapselt. Aufbau und Ablauf eines Codegeneratorsystems werden ebenfalls beschrieben.

Die durch den Codegenerator erstellte Client-Bibliothek wird anhand eines Anwendungsbeispieles evaluiert.

English

The present thesis describes the design of a codegenerator with exchangeable target-language, that generates a client-library from the abstract description of the Spreadshirt-API. The implementation of the codegenerator is made in Java, the target-language of the client-library is PHP.

The basics of web services, codegenerators and document-description-languages will be explained and based on that datamodels will be created, that contains the description of Spreadshirt-API, for the generator. Furthermore the created language-model will be examined, which encapsulates the constructs of the

target-language that will be generated. Structure and process of a codegeneratorsystem will be described, as well.

The client-library that was created by the codegenerator will be evaluated on the basis of a usage example.

Keywords

Codegeneration, RESTful Web Service, Modeling, Client-Library, Spreadshirt-API, Polyglot

Lizenz

Die vorliegende Bachelorarbeit **Generierung und Design einer Client-Bibliothek für einen RESTful Web Service am Beispiel der Spreadshirt-API** ist unter Creative Commons CC-BY-SA [1] lizenziert.

[1] http://creativecommons.org/licenses/by-sa/3.0/deed.de

Inhaltsverzeichnis

1 Einführung

»Essentially, all models are wrong, but some are useful.«

boxDraper
boxDraper *(***boxDraper***)*

Das Ziel dieser Arbeit ist die Erstellung eines Codegenerators, der aus der abstrakten Beschreibung der Spreadshirt-API eine Client-Bibliothek erstellt.

Der Generator soll eine flexible Wahl der Zielsprache bieten, wobei mit »Zielsprache« im folgenden die Programmiersprache der erzeugten Bibliothek gemeint ist. Für das Bibliotheksdesign ist eine DSL (Domain-Specific Language) zu realisieren, mit dem Ziel die Nutzung der API zu vereinfachen.

Als Programmiersprache für den Generator wird Java verwendet, PHP ist die Zielsprache der Bibliothek. Eine gute Lesbarkeit, hohe Testabdeckung und größtmögliche Typsicherheit, soweit PHP dies zulässt, sind Erfolgskriterien für die zu generierende Bibliothek.

Abbildung 1.1 stellt den schematischen Aufbau des gewünschten Generators dar.

Spreadshirt ist eines der führenden Unternehmen für personalisierte Kleidung und zählt zu den *Social Commerce*-Unternehmen. Dieser Begriff beschreibt Handelsunternehmen bei denen die aktive Beteiligung und die persönliche Be-

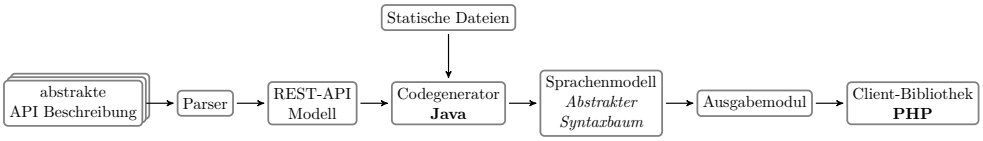

Abbildung 1.1: Aufbau des Generatorsystems

Abbildung 1.2: Spreadshirt Logo

ziehung sowie Kommunikation der Kunden untereinander im Vordergrund stehen. Spreadshirt hat Standorte in Europa und Nordamerika, der Hauptsitz ist in Leipzig.

Dem Nutzer wird eine Online-Plattform geboten um Kleidungsstücke selber zu gestalten oder zu kaufen, aber auch um eigene Designs, als Motiv oder in Form von Produkten, zum Verkauf anzubieten. Zusätzlich wird jedem Nutzer ermöglicht einen eigenen Shop auf der Plattform zu eröffnen und diesen auf der eigenen Internetseite einzubinden. Derzeit gibt es rund 400.000 Spreadshirt-Shops mit ca. 33 Millionen Produkten. Für die Spreadshirt-API können Kunden eigene Anwendungen schreiben, beispielsweise **zufallsshirt zufallsshirt** oder **soundslikecotton soundslikecotton** Spreadshirt bedient neben dem Endkunden- auch das Großkundengeschäft als Anbieter von Druckleistungen.

Die *sprd.net AG*, zu der auch der Leipziger Hauptsitz gehört, beschäftigt derzeit 450 Mitarbeiter, davon 50 in der IT.

Die zwei wichtigsten Konstanten in der Anwendungsentwicklung sind laut [**herrington2003code**] folgende:

- Die Zeit eines Programmierers ist kostbar.
- Programmierer mögen keine langweiligen und repetitiven Aufgaben.

Codegenerierung greift bei beiden Punkten an und kann zu einer Steigerung der *Produktivität* führen, die durch herkömmliches schreiben von Code nicht zu erreichen wäre.

Änderungen können an zentraler Stelle vorgenommen und durch die Generierung automatisch in den Code übertragen werden, was mit verbesserter *Wartbarkeit* und erhöhter Effizienz einhergeht. Die gewonnenen Freiräume kann

der Entwickler nutzen um sich mit den grundlegenden Herausforderungen und Problemen seiner Software zu beschäftigen.

Durch die Festlegung eines Schemas für Variablennamen und Funktionssignaturen wird eine hohe *Konsistenz* über die gesamte Codebasis hinweg erreicht. Diese Einheitlichkeit vereinfacht auch die Nutzung des Generats, da beispielsweise nicht mit Überraschungen bei den verwendeten Bezeichnern zu rechnen ist.

Zusätzlich zu dem bereits genannten allgemeinen Nutzen einer Codegenerierungslösung, entstehen für Spreadshirt noch die folgenden Vorteile:

- Vereinheitlichung bestehender Implementierungen in Form der generierten Bibliothek
- Kapselung der Authentifizierung durch Integration in Client-Bibliothek (**??**)
- Erleichterung der Api-Nutzung für externe Entwickler

1.1 Anforderungen an die Client-Bibliothek

- Austauschbarkeit der Zielsprache
- Einfache Bedienbarkeit der Bibliothek
- gute Lesbarkeit des erzeugten Codes
- größtmögliche Typsicherheit
- hohe Testabdeckung der Bibliothek
- vollständige Generierung aller Methoden aus der Api-Beschreibung

1.2 Typographische Konventionen dieser Bachelorarbeit

❶ -- ⓿: Für Verweise auf Elemente in Listings werden diese Symbole verwendet

Schlüsselwörter: Schlüsselwörter werden im Text *kursiv* hervorgehoben

`Quelltext`: Zur Darstellung von Quelltext wird eine `Konstantschrift` verwendet

2 Web Services

»The purpose of computing is insight, not numbers.«

Hamming
Hamming *(*Hamming*)*

In diesem Kapitel werden die Grundlagen zu HTTP, Dokumentbeschreibungssprachen, Beschreibungsformate für Webanwendungen und REST erläutert, welche für das Verständnis der Arbeit wichtig sind. Neben XML und JSON wird auch die Schemabeschreibungssprache XSD behandelt. Das Ende bildet die Einführung in REST und WADL.

2.1 Adressierung

Für den Zugriff auf Dienste eines Web Services werden Adressen benötigt. Adressen im Internet sind meist nach einem bestimmten Schema aufgebaut, üblicherweise sind dies URIs. Um Unklarheiten bei der Verwendung der Begriffe URI, URL und URN im Verlauf der Arbeit zu vermeiden, werden Sie an dieser Stelle definiert.

> **Definition 1** (URI)**.** *Ein »Uniform Resource Identifier« (*URI*) ist eine kompakte Zeichenkette zur Identifizierung einer abstrakten oder physischen Ressource. … Eine Ressource ist alles was identifizierbar ist, beispielsweise elektronische Dokumente, Bilder, Dienste und Sammlungen von Ressourcen. (eigene Übersetzung von* **w3cURI***).*

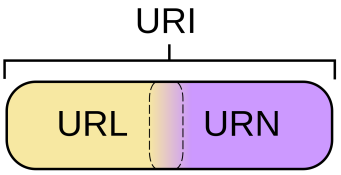

Abbildung 2.1: Diagramm zur Veranschaulichung der Teilmengenbeziehung zwischen den Adressierungsarten, Quelle **wiki:urn**

Definition 2 (URL)**.** *Der Begriff »Uniform Resource Locator« (*URL*) bezieht sich auf eine Teilmenge von* URIs*.* URLs *identifizieren Ressourcen über den Zugriffsmechanismus, anstelle des Namens oder anderer Attribute der Ressource. (eigene Übersetzung von* **w3cURI***).*

Definition 3 (URN)**.** *Eine Teilmenge der* URIs*, die sogenannten »Uniform Resource Names« (*URNs*), sind global eindeutige und beständige Bezeichner für Ressourcen. Sie müssen verfügbar bleiben auch wenn die bezeichnete Ressource nicht mehr erreichbar oder vorhanden ist. … Der Unterschied zu einer* URL *besteht darin, das ihr primärer Zweck in der dauerhaften Auszeichnung einer Ressource mit einem Bezeichner besteht. (eigene Übersetzung von* **w3cURI***).*

Beispiel URLs:

- `http://www.spreadshirt.net/`
- `mailto:admin@klingt.net`

Beispiel URNs:

- `urn:spreadshirt:product:2648`
- `urn:isbn:9780131392809`

2.2 HTTP

> **Definition 4** (HTTP)**.** *Das* Hypertext Transfer Protocol (HTTP) *ist ein allgemeines und* zustandsloses *Protokoll, zur Übertragung von Daten über ein Netzwerk, was durch Erweiterung seiner Anfragemethoden, Statuscodes und Header für viele unterschiedliche Anwendungen verwendet werden kann (***rfc2616***).*

HTTP arbeitet auf der Anwendungsschicht des OSI-Modell und ist somit unabhängig von dem zum Datentransport eingesetzten Protokoll. Über eindeutige URIs werden HTTP-Ressourcen angesprochen. Dabei sendet ein *Client* eine Anfrage (*request*) und erhält daraufhin vom Server eine Antwort (*response*). Anfrage und Antwort stellt dabei eine HTTP-Nachricht dar, die aus den zwei Elementen *Header* und *Body* besteht. Letzterer trägt die Nutzdaten und kann, je nach verwendeter HTTP-Methode, auch leer sein.

2.2.1 Methoden

rfc2616 (**rfc2616**) definiert einige HTTP-Methoden, wobei die am meisten verwendeten die folgenden sind:

- GET (Aufruf einer Webseite im Browser → GET-Request an Server)
- PUT
- POST (Übermittlung von Formulardaten eines Client an Server)
- DELETE
- OPTIONS

Alle bis auf POST und OPTIONS sind *idempotent* (**rfc2616** Kapitel 9), d.h. eine Komposition (Ausführung der Methoden hintereinander) der Methode führt zu demselben Ergebnis wie ein einzelner Aufruf.

Methodennamen sind üblicherweise Verben die die auszuführende Aktion beschreiben, deswegen werden HTTP-Methoden auch »verbs« genannt. Die Definition von eigenen Methoden ist möglich.

2.2.2 Header

Ein Header einer HTTP-Nachricht besteht aus einer *Request Line* (erste Zeile des Headers) und einer Menge von Schlüssel-Wert-Paaren. Listing 2.1 zeigt einen Beispiel-Header für die GET-Anfrage auf die Spreadshirt-API Ressource: `http://api.spreadshirt.net/api/v1/locales`.

```
1 GET ❶ /api/v1/locales ❷ HTTP/1.1 ❸
2 User-Agent: curl/7.29.0 ❹
3 Host: api.spreadshirt.net ❺
4 Accept: */* ❻
```

Listing 2.1: HTTP-Header von GET Request auf Spreadshirt-API Ressource `http://api.spreadshirt.net/api/v1/locales`

❶ Angabe der HTTP-Methode
❷ Ressource
❸ verwendete HTTP-Version
❹ *User-Agent*, Angabe zum Client-System, das die Anfrage versendet
❺ *Host*, der Server welcher die Anfrage erhält und der die Ressource ❷ verwaltet
❻ Angabe von *Content-Types*, die der Client als Antwort akzeptiert, in diesem Fall eine *Wildcard*, also alle Typen sind als Antwort erlaubt

Nachfolgend die *Response* auf den soeben beschrieben *Request*.

```
1  HTTP/1.1 200 OK ❶
2  Expires: Tue, 20 Aug 2013 19:05:25 GMT
3  Content-Language: en-gb
4  Content-Type: application/xml; ❷
5  charset=UTF-8 ❸
6  X-Cache-Lookup: MISS from fish07:80
7  X-Server-Name: mem1
8  True-Client-IP: 88.79.226.66
9  Date: Tue, 20 Aug 2013 07:20:25 GMT
10 Content-Length: 1659
11 Connection: keep-alive
```

Listing 2.2: HTTP-Header von GET Response aus Spreadshirt-API Ressource `http://api.spreadshirt.net/api/v1/locales`

❶ *Response Line*, Angabe der HTTP-Version am Anfang und danach der HTTP-Statuscode mit Kurzbeschreibung
❷ Angabe des Content-Types des Bodys der Nachricht (hier XML)
❸ Kodierung der Nachricht

Welche Einträge der Header einer HTTP-Nachricht letztendlich enthält, ist abhängig von der Implementierung des Clients oder Servers und es können auch jederzeit eigene Einträge, die nicht in der HTTP-Spezifikation enthalten sind, hinzugefügt werden.

2.2.2.1 Authorization Request Header

Die HTTP-Spezifikation **rfc2616** sieht ein Feld für Autorisierungsinformationen im Header vor. Der Feld ist folgendermaßen aufgebaut: `Authorization: credentials`. Der Aufbau des Berechtigungsnachweises (engl. »credentials«) ist nicht näher spezifiziert und kann vom Web Service Betreiber selbst festgelegt werden. Die Spreadshirt-API definiert einen eigenen Autorisierungs-Header (**??**).

2.2.3 Body

```
1  <?xml version="1.0" encoding="UTF-8" standalone="yes"?>
2  <locales
3      xmlns:xlink="http://www.w3.org/1999/xlink"
4      xmlns="http://api.spreadshirt.net"
5      xlink:href="http://api.spreadshirt.net/api/v1/locales"
6      offset="0" limit="50" count="16"
7      sortField="default" sortOrder="default">
8      <locale
9          xlink:href="http://api.spreadshirt.net/api/v1/locales/de_DE"
10         id="de_DE"/>
11     ...
12     <locale
13         xlink:href="http://api.spreadshirt.net/api/v1/locales/nl_BE"
14         id="nl_BE"/>
15 </locales>
```

Listing 2.3: HTTP-Body der Response aus der GET-Methode auf der Spreadshirt-API-Ressource `http://api.spreadshirt.net/api/v1/locales`

Der Body enthält die eigentlichen Nutzdaten. Deren Format wird mit dem *Content-Type*-Eintrag des Headers angegeben. Listing 2.3 zeigt den Body der *response* von Listing 2.2.

2.3 Dokumentbeschreibungsformate

In diesem Abschnitt werden die beiden von der Spreadshirt-API verwendeten Dokumentbeschreibungsformate, XML und JSON, behandelt. Außerdem wird die Schemabeschreibungssprache XML Schema Description (XSD) eingeführt.

2.3.1 XML

> **Definition 5** (XML)**.** *Die »Extensible Markup Language«, kurz* XML*, ist eine Auszeichnungssprache (»Markup Language«), die eine Menge von Regeln beschreibt um Dokumente in einem mensch- und maschinenlesbaren Format zu kodieren* **XML10Specification**

Obwohl das Design von XML auf Dokumente ausgerichtet ist, wird es häufig für die Darstellung von beliebigen Daten benutzt **wiki:xml** z.B. um diese für die Übertragung zu serialisieren.

Eine valide XML-Datei beginnt mit der XML-Deklaration ❶. Diese enthält Angaben über die verwendete XML-Spezifikation und die Kodierung der Datei. Im Gegensatz zu gewöhnlichen Tags wird dieses mit `<?` und mit `?>` beendet. Danach folgen beliebig viele baumartig geschachtelte *Elemente* mit einem Wurzelelement ❷. Die Elemente können Attribute enthalten und werden, wenn sie kein leeres Element sind ❸, von einem schließenden Tag in der gleichen Stufe abgeschlossen ❺. Nicht leere Zeichenketten als Kindelement sind ebenfalls erlaubt ❹.

Mit Hilfe von *Schemabeschreibungssprachen* (Abschnitt 2.4) kann der Inhalt und die Struktur eines Dokumentes festgelegt und gegen diese validiert werden. Der Begriff XML Schema ist mehrdeutig und wird oft auch für eine konkrete Beschreibungssprache, die »XML Schema Definition«, kurz XSD, verwendet.

```
1  <?xml version="1.0" encoding="UTF-8" standalone="yes"?> ❶
2  <design xmlns:xlink="http://www.w3.org/1999/xlink"
3          xmlns="http://api.spreadshirt.net"
4          ...>❷
5      <name>tape_recorder</name>
6      ...
7      <size unit="px">
8          <width>3340.0</width>
9          <height>3243.0</height>
10     </size>
11     <colors/>❸
12     ...
13     <created>
14         2013-03-30T12:37:54Z ❹
15     </created>
16     <modified>2013-04-02T11:13:02Z</modified>
17 </design>❺
```

Listing 2.4: Die gekürzte Antwort der API-Ressource `users/userid/designs/designID` als Beispiel für eine XML-Datei

2.3.2 JSON

Definition 6 (JSON)**.** *»JavaScript Object Notation«, kurz* JSON*, ist ein leichtgewichtiges, textbasiertes und sprachunbhängiges Datenaustauschformat. Es ist von JavaScript abgeleitet und definiert eine kleine Menge von Formatierungsregeln für die transportable Darstellung (Serialisierung) von strukturierten Daten (nach* **JSONRFC***).*

Im Gegensatz zu XML ist JSON weit weniger mächtig, es gibt z.B. keine Unterstützung für Namensräume und es wird nur eine geringe Menge an Datentypen unterstützt (Tabelle 2.1). Durch seine einfache Struktur wird aber ein deutlich geringerer »syntaktischer Overhead« erzeugt. Mit JSON-Schema **json-schema-draft** ist es möglich eine Dokumentstruktur vorzugeben und gegen diese zu validieren.

Objekte werden in JSON von geschweiften ❶, Arrays hingegen von eckigen Klammern begrenzt ❷. Objekte enthalten *key-value-pairs* (Schlüssel-Wert-

primitiv	strukturiert
Zeichenketten	Objekte
Ganz-, Fließkommazahlen	Arrays
Booleans	
null	

Tabelle 2.1: JSON Datentypen

Paare) ❸. Schlüssel sind immer Zeichenketten, die Werte dürfen von allen Typen aus Tabelle 2.1 sein und beliebig tief geschachtelt werden.

```
1  {
2      "name": "tape_recorder", ❸
3      "description": "",
4      "user": { ❶
5          "id": "1956580",
6          "href": "http://api.spreadshirt.net/api/v1/users/1956580"
7      }, ❶
8      "resources": [ ❷
9      ...
10         {
11             "mediaType": "png",
12             "type": "preview",
13             "href": "http://image.spreadshirt.net/image-server/v1/designs
                   /15513946"
14         },
15         ...
16     ], ❷
17     "created": "30.03.2013 12:37:54",
18     ...
19 }
```

Listing 2.5: Die gekürzte Antwort der API-Ressource `users/userid/designs/designID` als Beispiel für eine JSON-Datei

2.4 XML Schemabeschreibungssprachen (XML Schema)

Xml Schema bezeichnet Xml-basierte Sprachen, mit denen sich Elemente, Attribute und Aufbau einer Menge von Xml-Dokumenten — die dem Schema entsprechen — beschreiben lassen. Ein Xml-Dokument wird als *valid* gegenüber einem Schema bezeichnet, falls die Elemente und Attribute dieses Dokumentes die Bedingungen des Schemas erfüllen **taxonomyXMLSchema** Neben Xsd existieren noch weitere Schemasprachen. Diese sind für die Arbeit unwichtig und werden daher nicht behandelt.

2.4.1 XML Schema Description (XSD)

XML Schema Description ist ein stark erweiterte Nachfolger der Dtd (Document Type Definition), derzeit spezifiert in Version 1.1 **XMLSchema11Specification** Die Syntax von Xsd ist Xml. Damit ist die Schemabeschreibung ebenfalls ein gültiges Xml-Dokument. Als Dateiendung wird üblicherweise `.xsd` verwendet.

Die Hauptmerkmale von Xsd sind nach **taxonomyXMLSchema** die folgenden:

- komplexe Typen (strukturierter Inhalt)
- anonyme Typen (besitzen kein `type`-Attribut)
- Modellgruppen
- Ableitung durch Erweiterung oder Einschränkung (»derivation by extension/restriction«)
- Definition von abstrakten Typen
- Integritätsbedingungen (»integrity constraints«):
  unique, *keys* und *keyref*, dies entspricht den *unique-*, *primary-* und *foreign-*keys aus dem Bereich der Datenbanken

```
1  <?xml version="1.0" encoding="UTF-8" standalone="yes"?> ❶
2  <xs:schema xmlns:xs="http://www.w3.org/2001/XMLSchema" targetNamespace="http:
       //api.spreadshirt.net" version="1.0" elementFormDefault="qualified"> ❷
3      <xs:import namespace="http://www.w3.org/1999/xlink" schemaLocation="xlink.
           xsd"/> ❸
4      ...
```

Listing 2.6: Beginn der Xsd-Datei für die Spreadshirt-Api

Eine Xsd-Datei beginnt wie jede Xml-Datei mit der Xml-Deklaration ❶.

Das Wurzelelement der Schemadefinition zeigt ❷. Das Attribut *xmlns:xs= "http://www.w3.org/2001/XMLSchema"* führt den Namespace-Prefix *xs* ein und gibt außerdem an, dass die Elemente und vordefinierten Datentypen (**??**) aus dem Namensraum *http://www.w3.org/2001/XMLSchema* verwendet werden. Durch das Attribut *targetNamespace* wird der Namensraum der Elemente festgelegt, die in dieser Schemadefinition definiert werden. *Version* gibt die Xsd-Version an. Der Wert des Attributs *elementFormDefault* gibt an, ob Elemente des Schemas den *targetNamespace* explizit angeben müssen (»qualified«) oder ob dies implizit geschieht (»unqualified«), die Angabe ist optional.

Externe Schemadefinitionen lassen sich unter Angabe des Namensraumes und einer Uri zu der Xsd-Datei einbinden ❸.

Xml-Schema Description erlaubt die Definition von simplen Typen (»SimpleType«) und Typen mit strukturiertem Inhalt (»ComplexType«).

```
1  <xs:simpleType name="unit">
2      <xs:restriction base="xs:string"> ❶
3          <xs:enumeration value="mm"/> ❷
4          <xs:enumeration value="px"/> ❷
5      </xs:restriction>
6  </xs:simpleType>
```

Listing 2.7: Beispiel für einen SimpleType namens »unit« der Spreadshirt-API

SimpleType-Definitionen dienen zur Beschreibung einfacher Typen wie *Enumeratoren*, oder *Listen* für Daten eines primitiven Typs. Ein Beispiel für die Definition eines Enumerators durch einen SimpleType zeigt Listing 2.7. Der Basisdatentyp des Enumerators wird dabei durch die Angabe des Attributs *base* ❶ festgelegt. Zuordnung von Werten zu dem Enumerator zeigt ❷.

Durch einen SimpleType definierte Listen sind durch Leerzeichen separierte Strings, sie werden meist für den Wert eines Attributes einer XML-Datei verwendet.

```
1 <xs:simpleType name=colors>
2     <xs:list itemType="xs:string"/>
3 </xs:simpleType>
```

Listing 2.8: Beispiel für einen Listentyp definiert duch einen SimpleType

```
1 <test>red green blue</test>}
```

Listing 2.9: Beispielinstanz für Typ aus Listing 2.8

Die Definition eines strukturierten Typs zeigt Listing 2.10.

```
1  <xs:complexType name="abstractList" abstract="true"> ❶
2      <xs:sequence> ❷
3          <xs:element minOccurs="0" ❸ name="facets"> ❹
4              <xs:complexType> ❺
5                  <xs:sequence>
6                      <xs:element xmlns:tns="http://api.spreadshirt.net"
                           minOccurs="0" maxOccurs="unbounded" ref="tns:facet" ❻
                           />
7                  </xs:sequence>
8              </xs:complexType>
9          </xs:element>
10     </xs:sequence>
11     <xs:attribute xmlns:xlink="http://www.w3.org/1999/xlink" ref="xlink:href"/>
           ❼
12     <xs:attribute type="xs:long" name="offset"/> ❽
13     <xs:attribute type="xs:string" name="query"/>
14     ...
15 </xs:complexType>
```

Listing 2.10: Beispiel für eine Schemabeschreibung mit XSD anhand des »abstractList«-Typs der Spreadshirt-API

Das *ComplexType*-Tag ❶ umschließt die Definiton des strukturierten Typs. XML-Schema Description erlaubt das Definieren von abstrakten Typen, nur Ableitungen davon dürfen als Instanzen in einem Dokument auftreten. Abgeleitete Typen dürfen dabei den abstrakten Typ *erweitern* oder *einschränken* (»derivation by extension/restriction«).