HTWK Leipzig

Fakultät für Informatik, Mathematik & Naturwissenschaften

Bachelor-Thesis

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

Author:

Andreas Linz

10INB-T
admin@klingt.net
Leipzig, 25. Juli 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

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
Bachelor Thesis, HTWK-Leipzig, 25. Juli 2013

made with XeTeX, LaTeX and BibTeX.

Selbständigkeitserklärung

Ich erkläre hiermit, dass ich diese Bachelor-Thesis 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, 25. Juli 2013

Danksagungen

...

Abstract

Schlüsselwörter

Codegenerierung, RESTful Web Service, Modellierung, Client-Bibliothek, Spreadshirt-API, Polyglot

Lizenz

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

Inhaltsverzeichnis

Diese Seite wurde mit Absicht leer gelassen.

1 Einführung

„Essentially, all models are wrong, but some are useful.“

Empirical Model-Building and Response Surfaces. p. 424
***George E. P. Box, Norman R. Draper** (1987)*

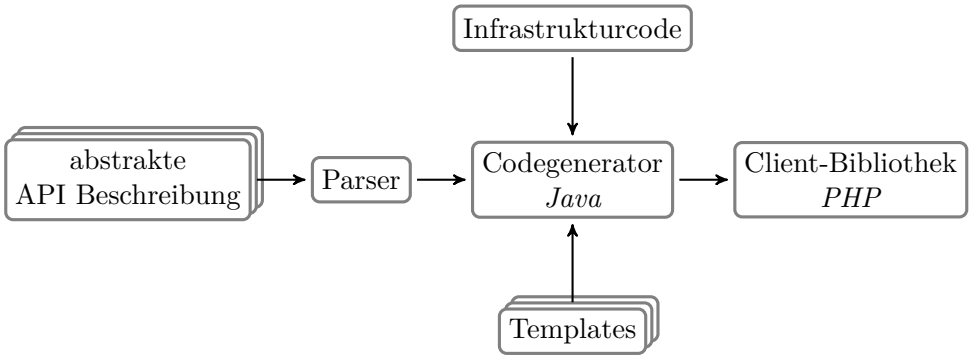

Abbildung 1.1: Aufbau des Generatorsystems

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, als Zielsprache der Bibliothek dient *PHP*. ~~Um die gewünschte Flexibilität bezüglich der Zielsprache zu erreichen, wird eine Template-Engine verwendet.~~ 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.

1.1 Was ist Spreadshirt?

Abbildung 1.2: Spreadshirt Logo

Spreadshirt ist eines der führenden Unternehmen für personalisierte Kleidung und zählt zu den *Social Commerce*[1]-Unternehmen. Es gibt Standorte in Europa und Nordamerika, der Hauptsitz ist in Leipzig. Den Nutzern wird eine Online-Plattform geboten um Kleidungsstücke selber zu gestalten oder zu kaufen, oder auch um eigene Designs, als Motiv oder in Form von Produkten, zum Verkauf anzubieten. Es wird jedem Nutzer ermöglicht einen eigenen Shop auf der Plattform zu eröffnen und ihn auf der eigenen Internetseite einzubinden. Derzeit gibt es rund 400.000 Spreadshirt-Shops mit ca. 33.000.000 Produkten. Für die Spreadshirt-API können Kunden eigene Anwendungen schreiben, bspw. zufallsshirt.de [2] oder soundslikecotton.com [3] . Neben dem Endkunden- bedient Spreadshirt auch das Großkundengeschäft als Anbieter von Druckleistungen.

Die *sprd.net AG* zu der auch der Leipziger Hauptsitz gehört beschäftigt derzeit[4] 178 Mitarbeiter, davon 29 in der IT.

[1] Handelsunternehmen bei dem die aktive Beteiligung und persönliche Beziehung, sowie Kommunikation der Kunden untereinander, im Vordergrund stehen.
[2] `http://zufallsshirt.de/`
[3] `http://www.soundslikecotton.com/`
[4] Stand Juni 2013

1.2 Motivation

Die zwei wichtigsten Konstanten in der Anwendungsentwicklung sind laut [Her03] 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* 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[5], da beispielsweise nicht mit Überraschungen bei den verwendeten Bezeichnern zu rechnen ist. Als Eingabe für den Generator dient ein *abstraktes Modell* des betreffenden Geschäftsbereiches. Die Erstellung eines solchen Modells vertieft das Verständnis des Entwicklers für das Geschäftsfeld und gibt gleichzeitig Spezialisten aus dem Fachbereich die Möglichkeit Fragestellungen anhand dieses Modells zu formulieren. Um die immer kürzeren Entwicklungszyklen einhalten zu können, kann durch Codegenerierung die nötige Effizienzsteigerung geleistet werden.

[5] Ergebnis des Codegenerierungsvorganges

2 Web Services

„The purpose of computing is insight, not numbers."

Numerical Methods for Scientists and Engineers. Preface
***Richard Hamming** (1962)*

Dieses Kapitel dient der Begriffserklärung und der Erläuterung weiterer Grundlagen die zum Verständnis dieser Arbeit notwendig sind. Es folgt eine Einführung in die beiden, von der Spreadshirt-API genutzten, Dokumentbeschreibungsformate XML und JSON. Speziell zu XML werden noch zwei Schemabeschreibungssprachen XSD und RelaxNG besprochen. Den Schluß bildet die Erläuterung von REST und des Webanwendungsbeschreibungsformates WADL.

2.1 Dokumentbeschreibungsformate

In diesem Abschnitt werden die Dokumentbeschreibungsformate XML und JSON, der Spreadshirt-API behandelt. Außerdem werden die Dokumentbeschreibungssprachen *XML Schema Description* und *RelaxNG* eingeführt.

2.1.1 XML

Definition 1 (XML)**.** *Die* Extensible Markup Language*, kurz XML, ist eine Auszeichnungssprache („Markup Language") die eine Menge von Regeln beschreibt um Dokumente in einem mensch- und maschinen lesbaren Format zu kodieren [W3C08].*

Obwohl das Design von XML auf Dokumente ausgerichtet ist, wird es häufig für die Darstellung von beliebigen Daten benutzt [Wik13c], z.B. um diese für die Übertragung zu serialisieren.

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

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* (siehe Abschnitt 2.2) 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.

2.1.2 JSON

Definition 2 (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 [Cro06]).*

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 (siehe Tabelle 2.1). Durch seine einfache Struktur wird aber ein deutlich geringerer „syntaktischen Overhead“ erzeugt. Mit JSON Schema [1] ist es möglich eine Dokumentstruktur vorzugeben und gegen diese zu validieren.

primitiv	**strukturiert**
Zeichenketten	Objekte
Ganz- und Fließkommazahlen	Arrays
Booleans	
null	

Tabelle 2.1: JSON Datentypen

Objekte werden in JSON von geschweiften- ❶, Arrays hingegen von geschlossenen Klammern begrenzt❷. Diese dürfen beliebig tief geschachtelt werden und auch Schlüssel-Wert-Paare (*key-value-pairs* ❸) enthalten. *Schlüssel sind immer Zeichenketten*, die Werte dürfen von allen Typen aus Tabelle 2.1 sein.

```
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         {
```

[1] http://tools.ietf.org/id/draft-zyp-json-schema-03.html

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

2.2 XML Schemabeschreibungssprachen (XML Schema)

XML Schema bezeichnet XML-basierte Sprachen mit denen sich Elemente, Attribute und Aufbau eines XML-Dokumentes beschreiben lassen. Ein XML-Dokument wird als *valid/gültig* gegenüber einem Schema bezeichnet, falls die Elemente und Attribute dieses Dokumentes die Bedingungen des Schemas erfüllen [Mur+05]. Neben XSD (siehe Abschnitt 2.2.1) und RelaxNG (siehe Abschnitt 2.2.2) existieren noch weitere Schemasprachen, die hier aber aufgrund ihrer geringen Relevanz nicht behandelt werden. Die beiden hier behandelten Schemasprachen bieten den Vorteil selbst XML-Dokumente zu sein, somit können sie durch herkömmliche XML-Tools bearbeitet werden.

2.2.1 XML Schema Description (XSD)

XML Schema Description ist ein stark erweiterte Nachfolger der *DTD* (Document Type Definition), derzeit spezifiert in Version 1.1 [W3C12]. 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 [Mur+05] , 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

Die XSD Spezifikation enthält bereits eine Menge vordefinierter Datentypen, dargestellt in Abbildung 2.1.

```
<?xml version="1.0" encoding="UTF-8" standalone="yes"?>
<xsd:schema
    xmlns:xsd="http://www.w3.org/2001/XMLSchema"
    version="1.0"
    targetNamespace="myNamespace"
    elementFormDefault="qualified">
    <xsd:complexType name="product">
        <xsd:sequence>
            <xsd:element name="name" type="xsd:string"/>
            <xsd:element name="price" type="xsd:decimal"/>
            <xsd:element name="description"
                ref="myNamespace:description"/>
        </xsd:sequence>
    </xsd:complexType>
    <xsd:complexType name="description">
        <xsd:all>
            <xsd:element name="title" type="xsd:string">
            <xsd:element name="content" type="xsd:string">
        </xsd:all>
    </xsd:complexType>
</xsd:schema>
```

Listing 2.3: Minimalbeispiel für eine Schemabeschreibung mit XSD

2.2.2 RelaxNG

Ebenso wie XSD (siehe Abschnitt 2.2.1) ist *Regular Language Description for XML New Generation* eine XML-Schemasprache zur Definition der Struktur von XML-Dokumenten. Schemas werden in *RelaxNG* durch XML-Syntax oder eine eigene, kompaktere nicht-XML Syntax formuliert. Ebenso wie bei *XML*

Schema werden Namespaces unterstützt. RelaxNG Schemabeschreibungen verwenden meist `.rng` als Dateiendung.

Unterschiede zu XML Schema:

- Unterstützung von ungeordneten Inhalten
- kompaktere nicht-XML Syntax
- *nichtdeterministisches* oder auch *mehrdeutiges* Inhaltsmodell ([Vli04] Kapitel 16)

```
<?xml version="1.0" encoding="utf-8"?>
<grammar
    xmlns="http://relaxng.org/ns/structure/1.0"
    ns="myNamespace">
    <start>
        <ref name="product"/>
    </start>
    <define name="product">
        <oneOrMore>
            <element name="name"/>
                <text/>
            </element>
            <element name="price"/>
                <text/>
            </element>
            <ref name="description"/>
        </oneOrMore>
    </define>
    <define name="description">
        <oneOrMore>
            <element name="title">
                <text/>
            </element>
            <element name="content">
                <text/>
            </element>
        </oneOrMore>
    </define>
</grammar>
```

Listing 2.4: Minimalbeispiel für eine Schemadefinition in RelaxNG

2.3 RESTful Web Service

Representational State Transfer (deutsch: „Gegenständlicher Zustandstransfer“) ist ein Softwarearchitekturstil für Webanwendungen, welcher von Roy Fielding[2] in seiner Dissertation aus dem Jahre 2000 beschrieben wurde [Kapitel 5 Fie00, 95 ff.].

Als RESTful bezeichnet man dabei eine Webanwendung die den Prinzipien von REST entspricht.

2.3.1 Elemente von REST

Im folgenden werden die Grundbausteine einer REST-Anwendung erläutert. Abschnitt 2.3.1.4 beschreibt die Komponenten, die an einer Aktion auf einer *Ressource* beteiligt sind. Dieser Abschnitt basiert auf Kapitel 5.2 von [Fie00, S. 86 ff.].

2.3.1.1 Ressource

Eine Ressource stellt die wichtigste Abstraktion von Information im REST-Modell dar. Fielding definiert eine *ressource* wie folgt:

> *„Any information that can be named can be a resource: a document or image, ….. A resource is a conceptual mapping to a set of entities, not the entity that corresponds to the mapping at any particular point in time.“*

Eine Ressource kann somit alle Konzepte abbilden die sich über einen Bezeichner referenzieren lassen. Dies können konkrete Dokumente, aber auch Dienste oder sogar Sammlungen von Ressourcen sein. Außerdem identifiziert ein Ressourcenbezeichner, meist eine *URI* (siehe Abschnitt 2.3.2.1), immer dieselbe Ressource, nicht aber deren Wert oder Zustand.

[2] Roy Thomas Fielding, geboren 1965, ist einer der Hauptautoren der HTTP-Spezifikation

2.3.1.2 Repräsentation

Eine Repräsentation (*representation*) stellt den aktuellen oder den gewünschten Zustand einer Ressource dar. Das Format der Repräsentation ist dabei unabhängig von dem der Ressource, siehe Abschnitt 2.3.2.4. Aktionen mit Komponenten einer REST-API werden durch den Austausch von solchen Repräsentationen durchgeführt. Im Allgemeinen wird unter einer Repräsentation nur eine Folge von Bytes verstanden, inklusive *Metainformationen* welche den Inhalt der Bytefolge klassifiziert, sowie *Kontrolldaten* die die gewünschte Aktion oder die Bedeutung der Anfrage beschreiben.

```
1       ...
2       <response>
3           <representation xmlns:sns="http://api.spreadshirt.net" ❶element=
                "sns:productTypes" status="200" ❷mediaType="application/xml"
                >
4               <doc title="Success"/>
5           </representation>
6           ...
7       </response>
```

Listing 2.5: Beispiel zu Metainformationen für REST-Repräsentation aus WADL-Datei der Spreadshirt-API

Ein Beispiel für eine solche Angabe von Metainformationen ist in Listing 2.5 zu finden, ❶ zeigt dies in Form einer *Typangabe* und eines *mediaType*-Attributes ❷.

Kontrolldaten sind meist HTTP-Header-Felder[3], bspw. um das Cachingverhalten zu ändern.

2.3.1.3 Konnektoren

Konnektoren stellen eine Schnittstelle für die Kommunikation mit Komponenten der REST-Webanwendung dar. Aktionen auf Ressourcen und der Austausch von Repräsentationen finden über diese Schnittstellen statt. Der Konnektor bildet die Parameter der Schnittstelle auf das gewünschte Protkoll ab.

[3] für die HTTP-Header-Definitionen siehe http://www.w3.org/Protocols/rfc2616/rfc2616-sec14.html#sec14.43

Eingangsparameter[4]:

- Anfrage-Kontrolldaten
- Ressourcenidentifizierung (Ressourcenbezeichner)
- · Repräsentation der Ressource

Ausgangsparameter:

- Antwort-Kontrolldaten
- · Metainformationen
- · Repräsentation der Ressource

Konnektor	Beispiel
client	libwww
server	libwww, Apache HTTP-Server API
cache	browser cache, Akamai
resolver	bind
tunnel	SOCKS

Tabelle 2.2: Beispiele für Konnektoren nach [Fie00]

2.3.1.4 Komponenten

Ursprungsserver:

Serverseitiger Konnektor der den Namespace der angeforderten Ressource verwaltet. Er ist die einzige Quelle für Representationen von und Änderungen auf seinen Ressourcen (siehe Abschnitt 2.3.2.1).

Proxy:

Zwischenkomponente die explizit vom Client verwendet kann, aus Sicherheits-, Performance- oder Kompatiblitätsgründen.

[4] · optionaler Parameter

Gateway:

Dient als Schnittstelle zwischen Client- und Servernetzwerk, kann zusätzlich aus den gleichen Gründen wie der Proxy verwendet werden. Konträr zum Proxy kann der Client aber nicht entscheiden ob er einen Gateway nutzen möchte.

User Client:

Ein clientseitiger *Konnektor*, der die Anfrage an die API startet und einziger Empfänger der Antwort ist. In den meisten Fällen ist dies einfach ein *Webbrowser*.

2.3.2 REST-Prinzipien

Die fünf Grundlegenden REST-Prinzipien, nach [Til09, 11 ff.]:

- Ressourcen mit eindeutiger Indentifikation
- Verknüpfungen / Hypermedia
- Standartmethoden[5]
- Unterschiedliche Repräsentationen
- Statuslose Kommunikation

2.3.2.1 Eindeutige Identifikation

Um eine *eindeutige Identifikation* zu erreichen, wird jeder Ressource eine URI vergeben. Dadurch ist es möglich zu jeder verfügbaren Ressource einen Link zu setzen. Nachfolgend eine Beispiel-URI, um den Artikel 42 aus dem Warenkorb 84 anzusprechen:

$$\underbrace{http://api.spreadshirt.net/api/v1/}_{Basis-URL}\overbrace{\underbrace{baskets/84}_{Warenkorb}\underbrace{/item/42}_{Artikel}}^{Ressource}$$

2.3.2.2 Hypermedia

Innerhalb einer Ressource kann auf weitere verlinkt werden (*Hypermedia*), als Nebeneffekt der eindeutigen Identifikation duch URIs sind diese auch außer-

[5] GET, PUT, POST, DELETE bei Nutzung von HTTP

halb des Kontextes der aktuellen Anwendung gültig. Das Folgen eines Links entspricht dabei einer Zustandsänderung innerhalb der Anwendung. Die vorhandenen Verknüpfungen legen fest welche Zustandsübergänge erlaubt sind.

2.3.2.3 Standartmethoden

Durch die Nutzung von *Standartmethoden* ist abgesichert das die Anwendung mit den Ressourcen arbeiten kann, vorausgesetzt sie unterstützt diese. REST ist nicht auf HTTP beschränkt, praktisch alle REST-APIs nutzen aber dieses Protokoll. HTTP umfasst dabei folgende Methoden[6]:

- GET
- PUT
- POST
- DELETE
- HEAD
- OPTIONS

Alle bis auf *POST* und *OPTIONS* sind *idempotent* ([Fie+99] Kapitel 9), d.h. eine hintereinander Ausführung der Methode führt zu demselben Ergebnis wie ein einzelner Aufruf.

2.3.2.4 Repräsentationen von Ressourcen

Die Repräsentation sollte unabhängig von der Ressource sein, um die Darstellung gegebenenfalls für den Client anzupassen. Per *Query-Parameter* oder als Information im HTTP-Header kann die Clientanwendung nun das gewünschte Format angeben und bekommt vom Server die entsprechend formatierte Antwort. Der Client kann anhand des *Content-Type* Feldes das Format der Antwort überprüfen, für JSON lautet dies bspw. `application/json`.

[6] Kapitel 9 des HTTP 1.1 RFC2616 beschreibt diese inklusive *TRACE* und *CONNECT* umfassend [Fie+99]

2.3.2.5 Statuslose Kommunikation

Es soll kein Sitzungsstatus (*session-state*) vom Server gespeichert werden, d.h. jede Anfrage des Client muss alle Informationen enthalten, die nötig sind um diese serverseitig verarbeiten zu können. Der Sitzungstatus wird dabei vollständig vom Client gehalten. Diese Restriktion führt zu einigen Vorteilen:

- Verringerung der Kopplung zwischen Client und Server
- zwei aufeinanderfolgende Anfragen können von unterschiedlichen Serverinstanzen beantwortet werden

↪ verbesserte Skalierbarkeit

Diese Vorteile werden mit erhöhter Netzwerklast erkauft, da die Statusinformationen bei jeder Anfrage mitgesendet werden müssen.

2.4 WADL

Die *Web Application Description Language* (kurz WADL) ist eine maschinenlesbare Beschreibung einer HTTP-basierten Webanwendung, einschließlich einer Menge von *XML Schematas* [Had06]. Die aktuelle Revision ist vom 31. August 2009 [7] , im weiteren beziehe ich mich aber hier auf die in der Spreadshirt-API verwendeten Version vom 9. November 2006[8].

Die Beschreibung eines Webservices durch WADL besteht nach [Had06] im groben aus den folgenden vier Bestandteilen:

Set of resources:
Analog einer Sitemap, die Übersicht aller verfügbaren Ressourcen

Relationships between resources:
Die kausale und referentielle Verknüpfung zwischen Ressourcen

Methods that can be applied to each resource:
Die von der jeweiligen Ressource unterstützten HTTP-Methoden, deren Ein- und Ausgabe, sowie die unterstützden Formate

[7] `http://www.w3.org/Submission/wadl/`
[8] Die Unterschiede zwischen beiden Revisionen können unter der folgenden URL nachvollzogen werden http://www.w3.org/Submission/wadl/#x3-41000D.1 [9] .

Resource representation formats:

Die unterstützten MIME-Typen und verwendeten Datenschemas (Abschnitt 2.2.1)

```
1  <?xml version="1.0" encoding="UTF-8" standalone="yes"?> ❶
2  <application xmlns="http://research.sun.com/wadl/2006/10"> ❷
3      <grammars> ❸
4          <include href="http://api.spreadshirt.net/api/v1/metaData/api.xsd
               ">
5              <doc>Catalog XML Schema.</doc>
6          </include>
7          ...
8      </grammars>
9      <resources base="http://api.spreadshirt.net/api/v1/"> ❹
10         <resource path="users/{userId}"> ❺
11             <doc>Return user data.</doc>
12             <method name="GET"> ❻
13                 <doc>...</doc>
14                 <request> ❼
15                     <param
16                         xmlns:xsd="http://www.w3.org/2001/XMLSchema"
17                         name="mediaType"
18                         style="query"
19                         type="xsd:string">
20                     <doc>...</doc>
21                     </param>
22                     ...
23                 </request>
24                 <response> ❽
25                     <representation
26                         xmlns:sns="http://api.spreadshirt.net"
27                         element="sns:user"
28                         status="200"
29                         mediaType="application/xml">
30                     <doc title="Success"/>
31                     </representation>
32                     <fault status="500" mediaType="text/plain">
33                         <doc title="Internal␣Server␣Error"/>
34                     </fault>
35                     ...
36                     }
37                 ...
```

Listing 2.6: Beispielaufbau einer WADL-Datei anhand der Spreadshirt-API Beschreibung

Die Datei beginnt mit der Angabe der XML-Deklaration ❶. Die Attribute des Wurzelknotens `<application>` enthalten *namespace* Definitionen, u. a. auch den der verwendeten WADL-Spezifikation ❷. Innerhalb des `<grammars>` Elements werden die benutzten *XML Schemas* angegeben ❸. Um die Ressourcen der Webanwendung ansprechen zu können wird noch die Angabe der Basisadresse benötigt ❹. Innerhalb des `<resources>` Elements findet sich Beschreibung der einzelnen Ressourcen. Diese sind gekennzeichnet, durch eine zur Basisadresse relativen URI ❺. In `{…}` eingeschlossene Teile einer URI, werden durch den Wert des gleichnamigen *request* Parameters ersetzt um die URI zu bilden (generative URIs). Im Folgenden werden die von der Ressource unterstützten HTTP-Methoden beschrieben ❻, deren Anfrageparameter `<request>` ❼, sowie die möglichen Ausgaben der Methode `<response>` ❽.

Die Dokumentations-Tags `<doc>` sind für alle XML-Elemente optional. Um das Listing nicht unnötig zu verlängern habe ich die schließenden *Tags* weggelassen.

Kapitel 2 von [Had06] beschreibt die Elemente im Detail.

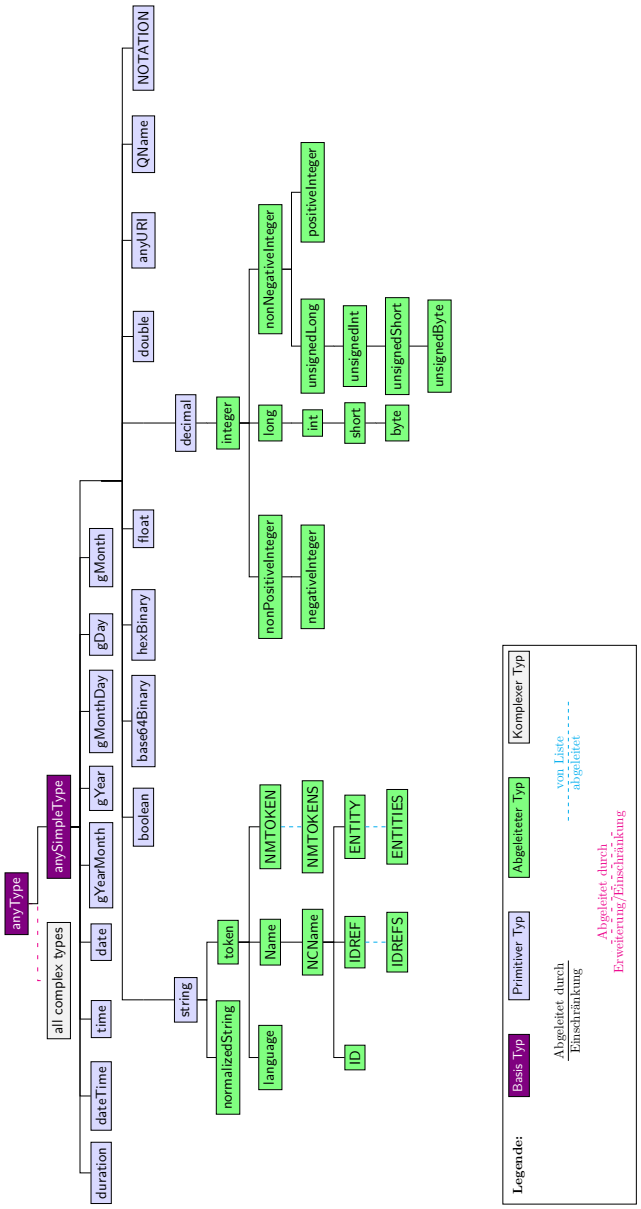

Abbildung 2.1: vordefinierte XSD Datentypen nach [W3C12] Kapitel 3

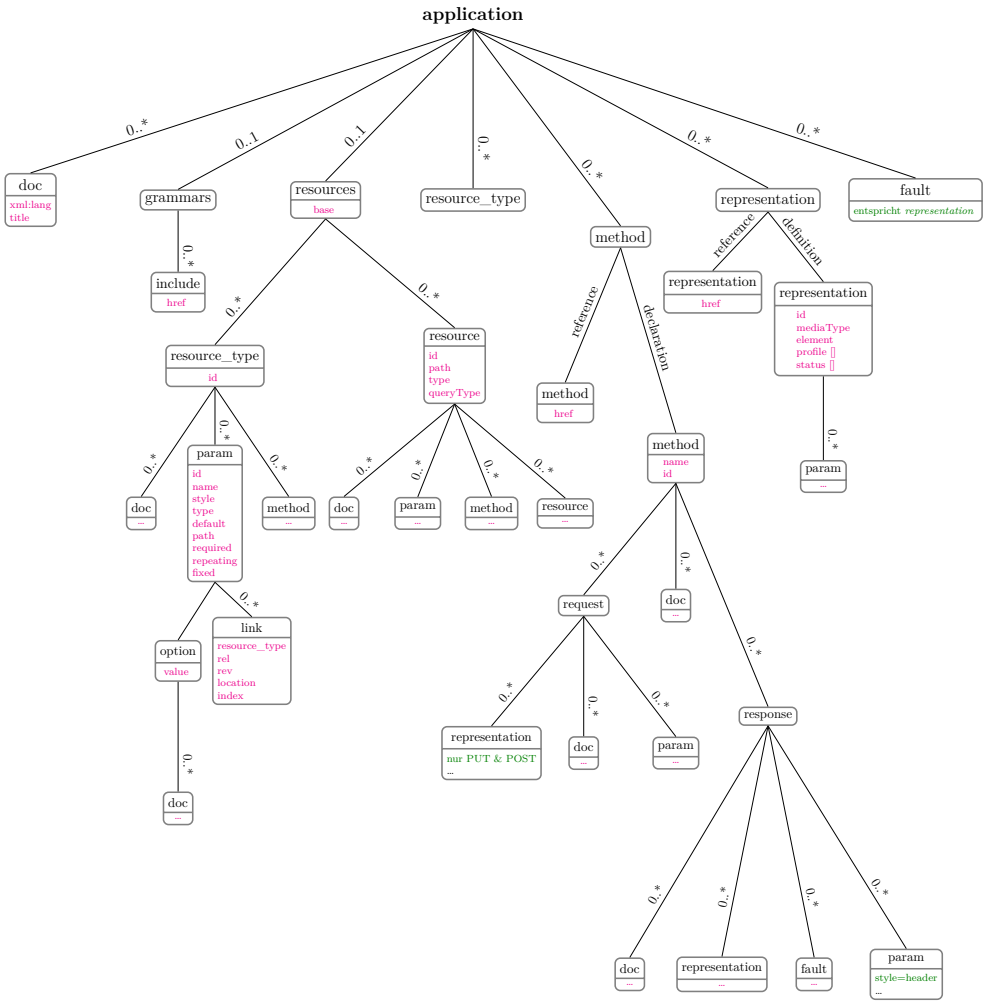

Abbildung 2.2: Struktur einer WADL-Datei, nach Kapitel 2 [Had06]

3 Datenmodell

Definition 3 (Datenmodell)**.** *Das Datenmodell, oder auch „Datenstruktur“ im Kontext von Programmiersprachen genannt, beschreibt die Beziehungen, Einschränkungen, Semantik und Struktur der darin enthaltenen Daten [Wik13a].*

Im Allgemeinen enthält das Modell die Daten von den Geschäftsprozessen gebraucht und erzeugt werden und erfasst explizit deren Struktur. Modelle werden meist in einer speziellen (grafischen) Notation beschrieben (bspw. ER-Diagramm).

Einteilung der Datenmodellschemas nach ANSI:

Konzeptuelles Schema:
: beschreibt die Semantiken der Domäne und legt den Gültigkeitsbereich des Datenmodells fest.

Logisches Schema:
: dient der Strukturbeschreibung des Modells. Wird oft durch Verfeinerung aus dem Konzeptuellen Schema entwickelt.

Physisches Schema:
: definiert die Form der Speicherung der Daten auf einem Physischen Datenträger.

Das Datenmodell dient als Eingabe für den Generator und ist somit die Basis für die Codegenerierung. Bei der Erstellung ist darauf zu achten das Modell möglichst präzise zu formulieren um Mehrdeutigkeiten bei der Interpretation zu vermeiden. Neben den direkt im Modell enthaltenen Informationen kann der Generator auch Abhängigkeiten darin erkennen und das Modell damit semantisch anreichern. Beispielweise die Beziehungen zwischen Datentypen die aus der XSD extrahiert wurden …

Das konkrete Modell für die Spreadshirt-API, setzt sich aus deren Beschreibung im WADL und XSD zusammen. Die Schnittpunkte beider Beschreibungen finden sich in der Angabe der Methodenparameter- und Rückgabewerte der WADL-Datei.

```
1  <resource path="users/{userId}/productTypes">
2      ...
3      <method name="GET">
4          ...
5          <request>
6              ...
7              <param xmlns:xsd="http://www.w3.org/2001/XMLSchema"❸ name="
                   sessionId" style="query" type="xsd:string" ❶>
8                  <doc>The security session id, e.g. 123.</doc>
9              </param>
10             ...
11         </request>
12         <response>
13             <representation xmlns:sns="http://api.spreadshirt.net"❹
                   element="sns:productTypes" ❷ status="200" mediaType="
                   application/xml">
14                 <doc title="Success"/>
15             </representation>
16             <fault status="500" mediaType="text/plain">
17                 <doc title="Internal␣Server␣Error"/>
18             </fault>
19             ...
20         </response>
21     </method>
22 </resource>
```

Listing 3.1: HTTP-GET-Methode zur Anzeige von Produkttypen eines Users (gekürzt)

Listing 3.1 zeigt einen Ausschnitt aus der Beschreibung der API-Methode zum Erhalt der *Produkttypen* eines bestimmten *Users*. Punkt ❶ und ❷ sind Typangaben, wobei ersteres einen Parameter- und letzteres einen Rückgabetyp ist. Die jeweiligen Namensräume der Typen werden unter ❸ und ❹ angegeben.

3.1 Abstract Syntax Tree (AST)

Definition 4 (Abstract Syntax Tree – Grune)**.** *Der abstrakte Syntaxbaum (engl. „Abstract Syntax Tree“) stellt die verschiedenen Teile eines Programmtextes aus Sicht der Grammatik, dar [GRB12, S. 9 ff.].*

Definition 5 (Abstract Syntax Tree – Aho)**.** *Ein Abstrakter Syntaxbaum ist die Darstellung eines Ausdrucks, wo jeder Knoten einen* Operator *und dessen Kindknoten die* Operanden *repräsentieren. Im Allgemeinen kann für jedes Programmierkonstrukt ein Operator erzeugt werden, dessen semantisch bedeutsamen Komponenten dann als Operanden gehandhabt werden (nach [Aho+06, S. 69]).*

Er ist das Endprodukt eines Parsingschrittes des Quelltextes, im Gegensatz zum *konkreten Syntaxbaum* (auch *Parse Tree*) enthält der *AST* keine Formatierungsspezifische Syntax (bspw. Klammern).

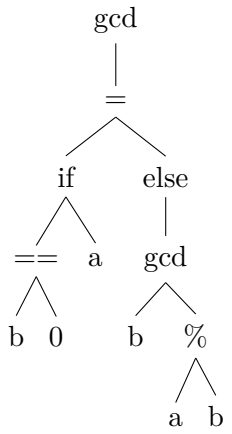

Abbildung 3.1: Beispiel AST für den rekursiven euklidischen Algorithmus

3.2 PHP Beispiele

```
1 <?php
2
```

```
3 ?>
```

Listing 3.2: Abfrage der Designs eines Users per HTTP-GET

4 Generatorsysteme

Definition 6 (Codegenerator)**.** *Ein* Codegenerator *ist ein Programm, welches aus einer höhersprachigen Spezifikation*[1] *einer Software oder eines Teilaspektes, die Implementierung erzeugt (nach [CE00]).*

Generatoren widmen sich drei wichtigen Problemen[CE00]:

Relevanz von Systembeschreibungen erhöhen:
Eine Systembeschreibung sollte direkt und explizit die Anforderungen bestimmen und mit der Sprache der Problemdomäne formuliert sein.

Erzeugung einer effizienten Implementierung:
Die größte Herausforderung bei der Erstellung eines Generators liegt in der Abbildung von der Spezifikation zur Implementierung, da es meist keine direkte Übereinstimmung zwischen beiden Konzepten gibt.

„Library scaling problem“:
Nur die durch die Spezifikation benötigten Methoden generieren.

Der Begriff „Generator“ ist sehr allgemein und wird für verschiedene Technologien verwendet, wie *Compiler*, *Präprozessoren*, Metafunktionen (Template-Metaprogramming in C++), Code-Transformatoren und natürlich Code-Generatoren.

4.1 Aufgaben eines Generators

1. Validieren der Spezifikation
2. Spezifikation durch Vorgabewerte vervollständigen
3. Optimierungen vornehmen

[1] m. a. W.: auf einem höheren Abstraktionslevel als die zur Implementierung verwendete Programmiersprache

4. Implementierung erzeugen

Je nach Form der Spezifikation, muss diese durch einen Analyse-Schritt (*parsing*) in die interne Darstellung des Generators überfürht werden, bspw. bei einem Compiler in einen *Abstrakten Syntaxbaum*. Der Informationsgehalt der Spezifikation ist ausschlaggebend für den Grad der zu erreichenden Automatisierung.

4.2 Nutzen einer Codegenerierungslösung für den Entwickler[Her03]

Qualität:
Bugfixes und Verbesserungen werden durch das Generatorsystem in die gesamte Codebasis propagiert.

Konsistenz:
Durch ein vorgegebenes Schema für die Schnittstellen- und Variablenbezeichner wird eine hohe Einheitlichkeit erreicht.

Zentrale Wissensbasis:
Das domänenspezifische Wissen wird in dem Metamodell gebündelt, das dem Generator als Eingabe dient. Änderungen am Modell werden durch den Generator in die gesamte Codebasis eingepflegt.

signifikantere Designentscheidungen:
Aufgrund des verringerten Implementierungsaufwandes kann der Entwickler mehr Zeit für das Design seiner Architektur , API etc. verwenden. Designfehlentscheidungen können durch Änderungen an den Templates korrigiert werden und bedürfen somit keiner manuellen Korrektur aller generierten Klassen.

Die Erstellung eines Generatorsystems geht mit einem nicht unerheblichen Aufwand einher, dieser sollte in Relation zum Umfang des zu erzeugenden Codes gesehen werden. Ist der Umfang des Erzeugnisses zu gering, kann der Aufwand zur Entwicklung einer Generatorlösung kontraproduktiv sein.

4.3 Generatorformen

Eine Möglichkeit zur Klassifikation von Codegeneratoren ist nach der Menge des erzeugten Codes:

teilweise	vollständig	mehrfach
Inline-Code Expander	Tier-Generator	n-Tier Generator
Mixed-Code Generator		
Partial-Class Generator		

Tabelle 4.1: Generatoren Klassifikation nach Generierungsmenge

Generatorformen nach [Her03]:

Inline-Code Expander:
Ein Inline-Code Expander nimmt Quellcode als Eingabe und ersetzt spezielle Mark-Up Sequenzen durch seine Ausgabe. Die Änderungen werden hierbei nicht in das Quellfile übernommen sondern meist direkt zu dem Compiler oder Interpreter weitergeleitet.

Mixed-Code Generator:
Der Mixed-Code Generator arbeitet grundsätzlich wie der Inline-Code Expander, seine Änderungen werden aber in die Quelle zurückgeschrieben.

Partial-Class Generator:
Partial-Class Generatoren erzeugen aus einer abstrakten Beschreibung und Templates einen Satz von Klassen, diese bilden aber nicht das vollständige Programm sondern werden durch manuell erzeugten Code vervollständigt.

Tier-Generator:
Die Arbeitsweise des Stufen- oder Tier[2]-Generators entspricht der des

[2] Tier, zu deutsch „Stufe“

Partial-Class Generators, mit der Ausnahme das vollständiger Code erzeugt wird, der keiner Nacharbeit bedarf.[3]

***n*-Tier Generator:**
Ein *n*-Tier Generator erzeugt neben dem eigentlichen Quellcode noch beliebige andere Informationen, bspw. eine Dokumentation oder Testfälle.

Die Entwicklung einer „Full-Domain Language“ stellt die oberste Stufe der Generatorformen. Eine solche Sprache ist Turing-vollständig und speziell auf die Problemdomäne ausgerichtet.

4.4 Optimierung durch den Generator

Die Effektivität von Optimierungen steigt mit dem Abstraktionslevel, deshalb ist es ratsam diese vom Generator durchführen zu lassen. Im Gegensatz zum Compiler, der viele dieser Optimierungen auch selbst durchführt, besitzt der Generator domänenspezifisches „Wissen“ (*domain-specific optimization*) und kann teilweise ohne Abhängigkeiten der Zielsprache optimieren.

Inlining:
Ein Symbol durch seine Definition ersetzen oder anstelle eines Funktionsaufrufes, die Deklaration der Funktion selbst einfügen

Constant folding:
Auswertung von Ausdrücken deren Operanden während der *compile time* bekannt sind.

Data caching:
Anstatt mehrfach denselben Ausdruck auszuwerten, das Ergebnis einmal berechnen und darauf an anderer Stelle referenzieren

Loop fusion:
Zusammenführen von Schleifen, die über den gleichen Bereich iterieren und deren Schleifenkörper unabhängig voneinander ist.

[3] Der im Laufe dieser Arbeit entwickelte Generator entspricht diesem Schema.

Loop unrolling:

Die Schleife durch n-mal deren Inhalt ersetzen, wobei n die Anzahl der Iterationen ist.

Code motion:

Invariante[4] Codebereiche aus dem Schleifenkörper herausnehmen

Dead-code elimination:

Entfernen von ungenutzten Variablen und unerreichbaren Codebereichen

Partial evaluation/Specialisation:

Partial evaluation oder auch *Specialisation* meinen das Erzeugen von spezialisierten Funktionen. Diese implementieren statische Eingaben mit dem Ziel einer kleineren Signatur.

Parallelization:

4.5 Konzeptuelles Modell

Definition 7 (Konzeptuelles Modell)**.** *Ein* Konzeptuelles Modell *beschreibt alle Objekte, Attribute, Rollen und Beziehungen sowie die Beschränkungen einer bestimmten Problemdomäne*[5]*. Bei der Erstellung des Modells werden explizit Design- und Implementierungsentscheidungen außer acht gelassen. Das Ziel ist die Bedeutung von Begriffen und Konzepten aus der Domäne zu formalisieren, Beziehungen zwischen den Konzepten finden und Mehrdeutigkeiten bei der Nutzung der Domänentermini zu vermeiden.*

Zur Darstellung werden häufig UML- oder ER-Diagramme verwendet, andere Formen sind ebenfalls möglich. Die Implementierung der Konzepte des Modells kann entweder manuell oder automatisch durch einen Generator erfolgen. Gerade letzteres ist wünschenswert, setzt aber eine große Sorgfalt bei der Erstellung des Modells voraus. Der Generator kann im Gegensatz zum

[4] unveränderliche

[5] Teil einer Applikation oder Fachbereich, der untersucht werden muss um die Problemstellung zu lösen. [Wik13b]

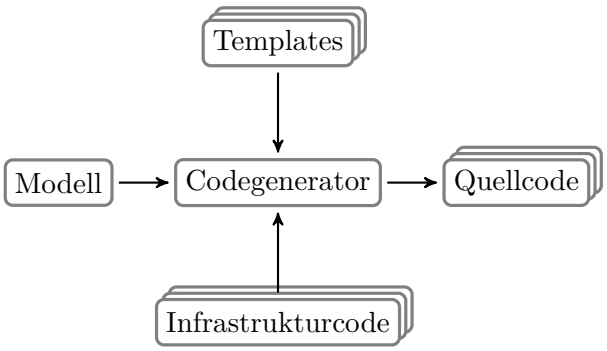

Abbildung 4.1: Simples Generatorsystem

menschlichen Entwickler keine eigenen Schlüße ziehen und muss daher mit den innerhalb des Modells enthaltenen Informationen auskommen.

4.6 Domain Specific Language (DSL)

Domain Specific Language (deutsch: „Domänenspezifische Sprache“) ist eine Programmiersprache die nur auf eine bestimmte Domäne oder auch Problembereich optimiert ist.

4.7 API-Design

5 Implementierung

5.1 XML-Parser

DOM-Parser vs. Streaming-Parser und Mittelding (StAX)

> *„ Streaming pull parsing refers to a programming model in which a client application calls methods on an XML parsing library when it needs to interact with an XML infoset—that is, the client only gets (pulls) XML data when it explicitly asks for it. Streaming push parsing refers to a programming model in which an XML parser sends (pushes) XML data to the client as the parser encounters elements in an XML infoset—that is, the parser sends the data whether or not the client is ready to use it at that time. “*

6 Evaluation

7 Zusammenfassung

7.1 Fazit

7.2 Ausblick

Glossar

Abstract Syntax Tree:
Ein *Abstrakter Syntaxbaum* ist die Baumdarstellung einer abstrakten Syntaktischen Struktur von Quellcode einer Programmiersprache. Jeder Knoten des Baumes kennzeichnet ein Konstrukt des Quellcodes. Der *AST* stellt für gewöhnlich nicht alle Details des Quelltextes dar, bspw. formatierende Element wie etwa Klammern werden häufig weggelassen. . A

API:
Application Programming Interface (deutsch: „Schnittstelle zur Anwendungsprogrammierung“) spezifiziert wie Softwarekomponenten über diese Schnittstelle miteinander interagieren können . 1, A

DSL:
Domain Specific Language (deutsch: „Domänenspezifische Sprache“) ist eine Programmiersprache die nur auf eine bestimmte Domäne oder auch Problembereich optimiert ist. . 1, A

DTD:
Document Type Definition, manchmal auch *Data Type Definition*, ist eine Menge von Angaben die einen Dokumenttyp beschreiben. Es werden konkret Element- und Attributtypen, Entitäten und deren Struktur beschrieben. Die bekanntesten Schemasprachen für XML-Dokumente sind XSD und RelaxNG. A, *siehe* XSD

GPL:
General Purpose Language bezeichnet eine Programmiersprache welche für die Entwicklung von Software aus einem großen Anwendungsbereich geeignet ist . A

JSON:
JavaScript Object Notation ist ein Mensch- und Maschinenlesbares Format zu Codierung und Austausch von Daten. Bietet im Gegensatz zu

XML keine Erweiterbarkeit und Unterstützung für Namesräume, ist aber kompakter und einfacher zu parsen. 4, 14, A, *siehe* XML

MDA:
Model Driven Architecture ist ein modell-getriebener Softwareentwicklungsansatz. Das zu modellierende System wird hierbei durch ein plattformunabhängiges Modell beschrieben mittels einer DSL beschrieben. Dieses Modell wird dann durch einen Generator in ein plattformspezifisches Modell, meist in einer GPL übersetzt . A

MDE:
Model Driven Engineering. A, *siehe* MDSD

MDSD:
Model Driven Software Development, auch *Model Driven Engineering* ist eine Softwareentwicklungsmethode welche ihren Fokus auf das erzeugen und nutzen von Domänen-Modellen, anstelle der algorithmischen Konzepte, legt . A

Metaprogramming:
beschreibt das erstellen von Programmen welche sich selbst, oder andere Programme, modifizieren oder die einen Teil des Kompilierungsschrittes übernehmen (bspw. der C-Präprozessor) . A

MIME:
Multipurpose Internet Mail Extensions dienen zu Deklaration von Inhalten (Typ des Inhalts) in verschiedenen Internetprotokollen. . 15, A

Polyglot:
mehrsprachig . A

RelaxNG:
Regular Language Description for XML New Generation ist ebenso wie *XSD* eine Schemabeschreibungssprache, bietet aber zwei Syntaxformen, eine XML basierte und eine kompaktere eigene Syntax. 4, A, *siehe* XSD

REST:
Representational State Transfer (deutsch: „Gegenständlicher Zustandstransfer“) ist ein Softwarearchitekturstil für Webanwendungen, welcher von Roy Fielding in seiner Dissertation [1] beschrieben wurde. Die Daten

[1] `http://www.ics.uci.edu/~fielding/pubs/dissertation/fielding_dissertation.pdf`

liegen dabei in eindeutig addressierbaren *resources* vor. Die Interaktion basiert auf dem Austausch von *representations* – also ein Dokument was den aktuellen oder gewünschten Zustand einer resource beschreibt. Beispiel-URL für das Item *84* aus dem Warenkorb *42*: `http://api.spreadshirt.net/api/v1/baskets/84/item/42` . 4, 10, 14, A

RESTful:
Als *RESTful* bezeichnet man einen Webservice der den Prinzipien von REST entspricht. 10, A, *siehe* REST

Template-Engine:
Eine *Template-Engine* ersetzt markierte Bereiche in einer Template-Datei (i. Allg. Textdateien) nach vorgegebenen Regeln . 1, A

URI:
Unified Resource Identifier ist ein Folge von Zeichen, die einen Name oder eine Web-Ressource identifiziert.. 13, 17, A

URL:
Unified Resource Locator sind eine Untermenge der *URIs*. Der Unterschied besteht in der expliziten Angabe des Zugrissmechanismus und des Ortes („Location“) durch *URLs*, bspw. `http` oder `ftp`. A, *siehe* URI

WADL:
Web Application Description Language ist eine maschinenlesbare Beschreibung einer HTTP-basierten Webanwendung. 4, 15, 21, A, *siehe* XML

XML:
Extensible Markup Language (deutsch: „erweiterbare Auszeichnungssprache“) ist ein Mensch- und Maschinenlesbares Format für Codierung und Austausch von Daten, spezifiziert vom W3C [2] . 4, A

XSD:
XML Schema Description, auch nur *XML Schema*, ist eine Schemabeschreibungssprache und enthält Regeln für den Aufbau und zum Validieren einer XML-Datei. Die Beschreibung ist selbst wieder eine gültige XML-Datei. 4, 5, 20, 21, A, *siehe* XML

[2] `http://www.w3.org/TR/REC-xml`

Abbildungsverzeichnis

Tabellenverzeichnis

Listings

Definitionsverzeichnis

Literaturverzeichnis

[Aho+06] Alfred V. Aho u. a. *Compilers: Principles, Techniques, and Tools (2nd Edition)*. Boston, MA, USA: Addison-Wesley Longman Publishing Co., Inc., 2006. ISBN: 0321486811.

[CE00] K. Czarnecki und U. Eisenecker. *Generative programming: methods, tools, and applications*. Addison Wesley, 2000. ISBN: 9780201309775. URL: http://books.google.de/books?id=cCZXYQ6Pau4C.

[Cro06] Douglas Crockford. *RFC 4627 - The application/json Media Type for JavaScript Object Notation (JSON)*. Techn. Ber. Juli 2006. URL: http://tools.ietf.org/html/rfc4627.

[Fie00] Roy Thomas Fielding. „Architectural styles and the design of network-based software architectures“. AAI9980887. Diss. 2000. ISBN: 0-599-87118-0.

[Fie+99] R. Fielding u. a. *Hypertext Transfer Protocol – HTTP/1.1*. United States, 1999.

[Fow10] M. Fowler. *Domain-Specific Languages*. Addison-Wesley Signature Series (Fowler). Pearson Education, 2010. ISBN: 9780131392809. URL: http://books.google.de/books?id=ri1muolw_YwC.

[GRB12] D. Grune, K. van Reeuwijk und H.E. Bal. *Modern Compiler Design*. Springer New York, 2012. ISBN: 9781461446996. URL: http://books.google.de/books?id=zkpFTBtK7a4C.

[Had06] Sun Microsystems Inc. Hadley Marc J. *Web Application Description Language (WADL)*. abgerufen am 21.06.2013. 9. Nov. 2006. URL: https://wadl.java.net/wadl20061109.pdf.

[Her03] J. Herrington. *Code Generation in Action*. In Action Series. Manning, 2003. ISBN: 9781930110977. URL: http://books.google.de/books?id=VHVC8WnSgbYC.

[KK06] M. Klar und S. Klar. *Einfach generieren: Generative Programmierung verständlich und praxisnah*. Hanser Fachbuchverlag, 2006. ISBN: 9783446404489. URL: http://books.google.de/books?id=6LS7OwAACAAJ.

[KT08] S. Kelly und J.P. Tolvanen. *Domain-Specific Modeling: Enabling Full Code Generation*. Wiley, 2008. ISBN: 9780470249253. URL: http://books.google.de/books?id=GFFtRFkuU_AC.

[Mur+05] Makoto Murata u. a. „Taxonomy of XML schema languages using formal language theory“. In: *ACM Trans. Internet Technol.* 5.4 (Nov. 2005), S. 660–704. ISSN: 1533-5399. DOI: 10.1145/1111627.1111631. URL: http://doi.acm.org/10.1145/1111627.1111631.

[Til09] Stefan Tilkov. *REST und HTTP: Einsatz der Architektur des Web für Integrationsszenarien*. Heidelberg: dpunkt, 2009. ISBN: 978-3-89864-583-6.

[Vli04] Eric van der Vlist. *RELAX NG - a simpler schema language for XML*. O'Reilly, 2004, S. I–XVIII, 1–486. ISBN: 978-0-596-00421-7.

[W3C08] W3C. *Extensible Markup Language (XML) 1.0 (Fifth Edition)*. 26. Nov. 2008. URL: http://www.w3.org/TR/2008/REC-xml-20081126/ (besucht am 25. 06. 2013).

[W3C12] W3C. *W3C XML Schema Definition Language (XSD) 1.1 Part 1: Structures*. 5. Apr. 2012. URL: http://www.w3.org/TR/2012/REC-xmlschema11-1-20120405/ (besucht am 30. 06. 2013).

[Wik13a] Wikipedia. *Data model — Wikipedia, The Free Encyclopedia*. [Online; accessed 17-July-2013]. 2013. URL: \url{http://en.wikipedia.org/w/index.php?title=Data_model&oldid=564342666} (besucht am 17. 07. 2013).

[Wik13b] Wikipedia. *Problem domain — Wikipedia, The Free Encyclopedia*. [Online; accessed 11-July-2013]. 2013. URL: \url{http://en.wikipedia.org/w/index.php?title=Problem_domain&oldid=544785678} (besucht am 11. 07. 2013).

[Wik13c] Wikipedia. *XML — Wikipedia, The Free Encyclopedia*. [Online; accessed 26-June-2013]. 2013. URL: \url{http://en.wikipedia.org/w/index.php?title=XML&oldid=561587115} (besucht am 26. 06. 2013).

BibTeX Eintrag

```
@phdthesis{AndreasLinz2013,
        type = {Bachelorthesis}
        author = {Linz, Andreas},
        year = {2013},
        month = {August},
        timestamp = {20130831},
        title = {Generierung und Design einer Client-Bibliothek für einen
           RESTful Web Service am Beispiel der Spreadshirt-API},
        school = {HTWK-Leipzig},
        pdf = {ToDo: PUT IN THE URL}
}
```