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, 1. 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, 1. 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, 1. 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)*

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 mit 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[1], 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

[1] Ergebnis des Codegenerierungsvorganges

immer kürzeren Entwicklungszyklen einhalten zu können, kann durch Codegenerierung die nötige Effizienzsteigerung geleistet werden.

1.1 Was ist Spreadshirt?

Abbildung 1.1: Spreadshirt Logo

Spreadshirt ist eines der führenden Unternehmen für personalisierte Kleidung und zählt zu den *Social Commerce*[2]-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 [3] oder soundslikecotton.com [4] . 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[5] 178 Mitarbeiter, davon 29 in der IT.

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

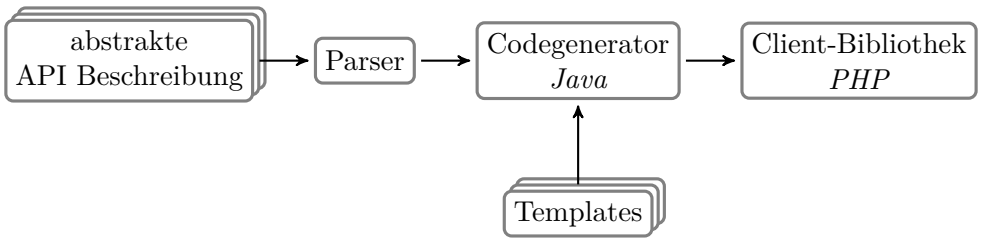

Abbildung 1.2: Aufbau des Generatorsystems

1.2 Ziel der Arbeit

Es ist ein Codegenerator zu erstellen, der aus der abstrakten Beschreibung der RESTful 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.

2 Grundlagen

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

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

Dieses Kapitel dient der Begriffsklärung und der Erläuterung weiterer Grundlagen die zum Verständnis dieser Arbeit notwendig sind. Es folgt eine Einführung in REST sowie der Beschreibungsformate WADL und XSD. Außerdem werden verschiedene Codegenerierungsformen vorgestellt.

2.1 Dokumentbeschreibungsformate

Der folgende Abschnitt soll eine kurze Einführung in die zwei am meisten verwendeten und von der Spreadshirt-API unterstützten, Formate für Datenaustausch geben. Außerdem wir ein Einblick in die XML Strukturbeschreibungssprache XML Schema gegeben.

2.1.1 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 [Wik13], z.B. um diese für die Übertragung zu serialisieren.

Mit Hilfe von *XML Schema* (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.

```
1 <?xml version="1.0" encoding="UTF-8" ?> ❶
2 <tagname key="value"> ❷
3     <emptyElement i_am_empty="true"/> ❸
4     <person>
5         <surname>Andreas</surname>
6         <lastname>Linz</lastname>
7     </person>
8 </tagname> ❹
```

Listing 2.1: Minimalbeispiel 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 könnten Attribute enthalten und werden, wenn sie kein leeres Element sind ❸, von einem schließenden Tag in der gleichen Stufe abgeschlossen ❹. Leere Elemente dürfen Attribute enthalten.

2.1.2 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 2.2). Andererseits erzeugt es bei der Serialisierung, durch seine einfache Struktur, einen deutlich geringeren „syntaktischen Overhead“. Mit JSON Schema [1] ist es möglich eine Dokumentstruktur vorzugeben und gegen diese zu validieren.

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

primitiv	strukturiert
Zeichenketten	Objekte
Ganz- und Fließkommazahlen	Arrays
Booleans	
null	

Tabelle 2.2: JSON Datentypen

```
{
        "key": "value",
        "another_key": 42,
        "an_object": {
                "foo": "bar",
                "some_boolean": false
        },
        "an_array": [
                "surname": "Andreas",
                "lastname": "Linz"
        ],
        "empty": null
}
```

Listing 2.2: Minimalbeispiel für eine JSON-Datei

2.2 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.

2.2.1 XML Schema Description (XSD)

Die *XML Schema Description* ist der 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. Die Spreadshirt API liefert XSDs zur Schemabeschreibung aus, deshalb wird die Beschreibungssprache im folgenden detailierter behandelt.

Komplexe Typen werden durch Elemente vom Typ `xsd:complexType` definiert, sie dienen zur Definition von XML-Inhalt aus Elementen mit Attributen. Elemente können hierbei Deklarationen oder Referenzen auf Elementdeklarationen sein. Simple Typen `xsd:simpleType`

2.2.2 RelaxNG

Regular Language Description for XML New Generation ist eine XML-Schemasprache zur Definition und Struktur von XML-Dokumenten. Schemas werden in *RelaxNG* durch XML-Syntax oder durch eine eigene, kompaktere nicht-XML Syntax formuliert. Ebenso wie bei *XML Schema* werden Namespaces unterstützt.

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 [Fie00].

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

Die fünf Grundlegenden REST-Prinzipien (nach [Til09]):

- Ressourcen mit eindeutiger Indentifikation
- Verknüpfungen / Hypermedia
- Standartmethoden[3]

[2] Roy Thomas Fielding, geboren 1965, ist einer der Hauptautoren der HTTP-Spezifikation
[3] GET, PUT, POST, DELETE bei Nutzung von HTTP

- Unterschiedliche Repräsentationen
- Statuslose Kommunikation

2.3.1 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:

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

2.3.2 Hypermedia

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

2.3.3 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[4]:

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

[4] Kapitel 9 der HTTP 1.1 Spezifikation beschreibt diese umfassend [Gro97]

Alle bis auf *POST* sind *idempotent*, d.h. eine hintereinander Ausführung der Methode führt zu demselben Ergebnis wie ein einzelner Aufruf.

2.3.4 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`.

2.3.5 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 [5] , im weiteren beziehe ich mich aber hier auf die in der Spreadshirt-API

[5] `http://www.w3.org/Submission/wadl/`

verwendeten Version vom 9. November 2006[6].

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

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

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

```
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 2.3: 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.

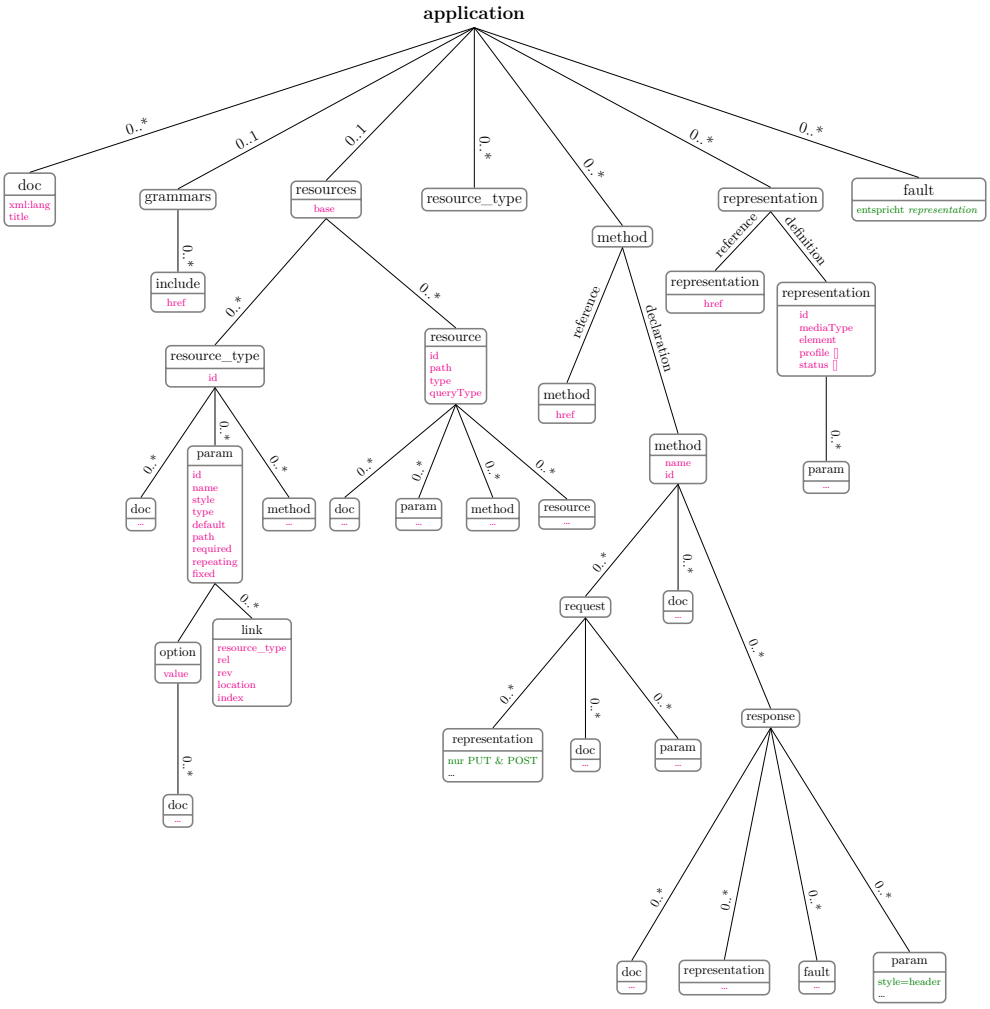

Abbildung 2.1: Struktur einer WADL-Datei, nach Kapitel 2 [Had06]

3 Implementierung

4 Zusammenfassung

4.1 Fazit

4.2 Ausblick

Glossar

API
: *Application Programming Interface* (deutsch: „Schnittstelle zur Anwendungsprogrammierung“) spezifiziert wie Softwarekomponenten über diese Schnittstelle miteinander interagieren können . 3, A

DSL
: *Domain Specific Language* (deutsch: „Domänenspezifische Sprache“) ist eine Programmiersprache die nur auf eine bestimmte Domäne oder auch Problembereich optimiert ist. . 3, 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

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. 9, A, *siehe* XML

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. . 10, A

Polyglot
: *mehrsprachig* . A

REST

Representational State Transfer (deutsch: „Gegenständlicher Zustandstransfer“) ist ein Softwarearchitekturstil für Webanwendungen, welcher von Roy Fielding in seiner Dissertation [1] beschrieben wurde. Die Daten 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, 7, 8, A

RESTful

Als *RESTful* bezeichnet man einen Webservice der den Prinzipien von REST entspricht. 3, 7, A, *siehe* REST

Template-Engine

Eine *Template-Engine* ersetzt markierte Bereiche in einer Template-Datei (i. Allg. Textdateien) nach vorgegebenen Regeln . 3, A

URI

Unified Resource Identifier ist ein Folge von Zeichen, die einen Name oder eine Web-Ressource identifiziert.. 8, 12, 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, 9, 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

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

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

XSD

XML Schema Description enthält Regeln für den Aufbau und zum Validieren einer XML-Datei. 4, 5, A, *siehe* XML

Abbildungsverzeichnis

Tabellenverzeichnis

Listings

Literaturverzeichnis

[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.

[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.

[Gro97] Network Working Group. *Hypertext Transfer Protocol – HTTP/1.1*. Jan. 1997. URL: http://www.w3.org/Protocols/rfc2068/rfc2068 (besucht am 25. 06. 2013).

[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.

[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).

[Wik13] 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}.

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}
}
```